Open Source Law, Policy and Practice

Open Source Law, Policy and Practice

Second Edition

Edited by

AMANDA BROCK

OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS

Great Clarendon Street, Oxford, OX2 6DP,
United Kingdom

Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship,
and education by publishing worldwide. Oxford is a registered trade mark of
Oxford University Press in the UK and in certain other countries

First Edition published in 2013
Second Edition published in 2022

Impression: 1

Published in the United States of America by Oxford University Press
198 Madison Avenue, New York, NY 10016, United States of America

British Library Cataloguing in Publication Data

Data available

Library of Congress Control Number: 2021946683

ISBN 978–0–19–886234–5

DOI: 10.1093/oso/9780198862345.001.0001

Printed and bound by
CPI Group (UK) Ltd, Croydon, CR0 4YY

Oxford University Press would like to thank the following organisations and/or individuals for granting permission for the use of their logos for the cover of this book:

All Things Open (‘All Things Open’ logo)

Python Software Foundation (‘Python’ and the Python logos are trademarks or registered trademarks of the Python Software Foundation, used by Oxford University Press with permission from the Foundation.)

Docker, Inc. (Docker and the Docker logo are trademarks or registered trademarks of Docker, Inc. in the United States and/or other countries.)

GNOME Foundation (The GNOME logo and GNOME name are registered trademarks or trademarks of GNOME Foundation in the United States or other countries.)

Rust Foundation

Vietsch Foundation (‘Vietsch Foundation’ logo)

Open Source Initiative (The OSI logo trademark is the trademark of Open Source Initiative)

Percona, LLC (‘Percona’ logo)

Nextcloud GmbH (Nextcloud and the Nextcloud Logo is a registered trademark of Nextcloud GmbH. in Germany and/or other countries.)

Open Innovation Network LLC (‘Open Innovation Network’ logo)

Tidelift, Inc. (‘Tidelift’ logo)

Canonical Limited (‘Ubuntu’ logo - Canonical Limited, Ubuntu, London)

Weaveworks, Inc. (‘Weaveworks’ logo - (c) 2014-2022 Weaveworks, Inc. All rights reserved.)

The Linux Foundation (The ‘Kubernetes’ logo is a registered trademark of The Linux Foundation)

Larry Ewing (‘The Tux’ and ‘The GIMP’ logos)

OpenUK (‘OpenUK’ logo)

This book has been a labour of love and is dedicated to the boys I love—Ronan, Rhys, and Dundee—and to the memory of my father, Chick, for whom I spent twenty-five years being a lawyer.

Contents

PART 1 INTELLECTUAL PROPERTY, CORPORATE, AND GOVERNANCE

PART 2 THE BUSINESS OF OPEN: ECONOMICS, OPEN SOURCE MODELS, AND USAGE

PART 3 EVERYTHING OPEN

Foreword

Keith Bergelt, CEO, Open Invention Network

Open Source software is the single-most impactful driver of innovation in the world today. The fact that it is a social movement supporting the notion of collaborative development cross-sector, cross-industry, and among and between individuals of different nationalities, races, and religions allows it to serve as an enduring model for innovation. No longer is software being developed in corporate silos where there is a cap on innovative output. By bringing smart people together from diverse backgrounds and experiences, elegant and functional code is being produced that would otherwise not be accessible.

While this model may have initially drawn adherents from primarily Western Europe and the United States, Japanese and Korean companies and individual coders began to participate actively in Open Source software projects in the mid-late 2000s and have been closely followed by Chinese company participants over the last eight to ten years. In fact, recognition of the inevitability of Open Source has resulted in global participation in Open Source software projects managed by the likes of the Apache Foundation, Eclipse Foundation, the Linux Foundation, and many other organisations that have emerged to provide professional project management and ensure an efficient path to the release of important code that can be freely adopted and around which innovative products can be cost-effectively produced.

As Open Source software has evolved and proliferated in information technology (IT), telecommunications, electronics, mobile communications, computing, transportation, energy, banking, financial services, fintech, big data, the Internet of Things (IoT), and many other sectors, the need for knowledgeable and experienced legal counsel has become acute. Copyright, trademark, and patent attorneys, in parallel with the explosive level of technical collaboration in Open Source software project communities, have been working in networks such as those managed and maintained by the Free Software Foundation Europe (European Legal Network), Linux Foundation (Member Legal Council), and Open Invention Network (Asian Legal Network) to share best practices and accelerate Open Source community-wide legal proficiency, and through journals like the *Journal of Law Technology and Society* (formerly *Free and Open Source Software Law Review*).

Open Source software projects such as the Software Project Data Exchange (SPDX) and OpenChain, both explored in this text, have emerged as ISO-approved standards to enable content management and process discipline that ensures copyright compliance as part of comprehensive governance programs. Software compliance tool companies have also emerged to further support active copyright compliance.

On the patent front, Open Invention Network manages an ever-growing 3700-strong network of the largest and most significant patent holding companies in the world, committed to cross-licensing each others' patents that read on core Linux and adjacent Open Source functionality and, in the process, forebearing patent infringement litigation. In addition, IBM, Microsoft, and the Linux Foundation have joined with Open Invention Network to found the Unified Patents' Open Source Zone and mitigate patent risk to the broader Open Source community posed by patent assertion entities.

The recurring theme across the Open Source technical and legal communities is that of collaboration.

Individuals and organisations come together to yield new novelty and innovate at unprecedented levels. Lawyers, recognising the need to build a community to protect and nurture the integrity of the social movement that underlies Open Source technical development, collaborate to enable copyright compliance and patent risk mitigation in the core of Linux and Open Source project functionality and generously share their knowledge.

At the end of the day, Open Source is about opportunity and obligation whereby manifest across the community is an implicit understanding that the opportunity to enjoy the benefits of co-opetition through Open Source project participation requires a concomitant obligation to adhere to a code of legal conduct and set of social norms.

Abbreviations

ACTA	Anti-Counterfeiting Trade Agreement
AGPL	GNU Affero General Public Licence
AI	artificial intelligence
AIA	America Invents Act
AOSP	Android Open Source Project
API	application programming interface
ASF	Apache Software Foundation
ASP	application service provider
AST	Allied Security Trust
BD	benevolent dictator
BIOS	Basic Input/Output System
BIS	Bureau of Industry and Security
BOLO	Be on the Look Out
BOM	Bill of Materials
BSD	Berkeley Software Distribution
BSL	Business Source Licence
CAD	computer-aided design
CAL	Cryptographic Autonomy Licence
CC	Creative Commons
CCBY	Creative Commons Attribution Licence
CCL	Confluent Community Licence
CC0 1.0	Creative Commons Universal Public Domain Dedication
CCS	Crown Commercial Service
CDDL	Common Development and Distribution Licence
CEO	Chief Executive Officer
CI/CD	Continuous Integration/Continuous Development
CII	computer-implemented inventions
CIO	Chief Information Officer
CLA	contributor licence agreement
CNC	computer numerical control
CNCF	Cloud Native Computing Foundation
CONTU	Commission on New Technological Uses of Copyrighted Works
COSS	commercial Open Source software
COTS	Commercially available off-the-shelf
CPDA	Copyright Designs and Patents Act 1988
CPP	C++
CSIS	Center for Strategic and International Studies
CSV	comma-separated values

CTO	Chief Technical Officer
DAO	Decentralized Autonomous Organization
DCO	Developer's Certificate of Origin
DD	Debian Developers
DFARS	Defense Federal Acquisition Regulations
DLT	Distributed Ledger Technology
DMCA	Digital Millennium Copyright Act
DOAJ	Directory of Open Access Journals
DoD	Department of Defense
DPL	Debian Project Leader
DRM	Digital Rights Management
DVD	digital video disc
EAR	Export Administration Regulations
ECJU	Export Control Joint Unit
ECtHR	European Court of human Rights
ECJ	European Court of Justice
EFF	Electronic Freedom Foundation
ENC	Environmental Noise Cancellation
ENT	Espace Numérique de Travail
EPC	European Patent Convention
EPL	Eclipse Public Licence
EPO	European Patent Office
EU	European Union
EUPL	European Public License
FAQs	frequently asked questions
FARS	Federal Acquisition Regulations
FDL	Free Documentation Licence
FLA	Fiduciary Licence Agreement
FLOSS	Free Libre and Open Source Software
FMCG	fast-moving consumer goods
FPGA	field programmable gate array
FRAND	fair, reasonable, and non-discriminatory
FSD	Free Software Definition
FSF	Free Software Foundation
FSFE	Free Software Foundation Europe
FTC	Federal Trade Commission
FTP	File Transfer Protocol
FUD	fear, uncertainty, and doubt
GATS	General Agreement on Trade in Services
GATT	General Agreement on Tariffs and Trade
GCC	GNU C++ Compiler
GDP	gross domestic product
GDPR	General Data Protection Regulation
GDS	Government Digital Service
GEA	General Export Authorisation

GPA	Agreement on Government Procurement
GPL	General Public Licence
HDL	hardware description language
HP	Hewlett Packard
ICO	Initial Coin Offering
ICT	information and communications technology
IDABC	Interoperable Delivery of European eGovernment Services to Public Administrations, Businesses and Citizens
IEA	International Energy Agency
IEC	International Electrotechnical Commission
IETF	International Engineering Task Force
IoT	Internet of Things
IP	intellectual property
IPO	Initial Public Offering
IP	intellectual property right
IPR	Inter Partes Review
IRS	Internal Revenue Service
ISO	International Organization for Standardization
IT	information technology
ITAR	International Traffic in Arms Regulations
ITC	International Trade Court
ITU	International Telecommunication Union
JEDEC	Joint Electron Device Engineering Council Standards Development Organisation
KDE	K Desktop Environment
LAMP	Linux, Apache, MySQL, PHP
LFCF	Linux Foundation Climate Finance Foundation
LGPL	GNU Lesser Public Licence
LKM	loadable kernel module
LoC	lines of code
LOT	Licence on Transfer
M&A	mergers and acquisitions
MNO	Mobile Network Operator
MOFCOM	Ministry of Commerce (China)
MOST	Ministry of Science and Technology (China)
MPL	Mozilla Public Licence
NASA	National Aeronautics and Space Administration
NC	Creative Commons Non-commercial
NCSA	The National Cyber Security Alliance
ND	no derivatives
NDA	non-disclosure agreement
NHS	National Health Service
NIST	National Institute of Standard and Technology
NPEs	non-practising entities
NTIA	National Telecommunications and Information Administration

OASIS	Organization for the Advancement of Structured Information Standards
OCI	Open Container Initiative
ODH	openly designed hardware
ODI	Open Data Institute
OECD	Organisation for Economic Co-operation and Development
OEM	original equipment manufacturer
OFAC	Office of Foreign Assets Control
OGEL	open general export licence
OIN	Open Invention Network
OKF	Open Knowledge Foundation
on-prem	on premises
OS	Operating System
OSD	Open Source Definition
OSI	Open Source Initiative
OSL	Open Software License
OSPO	Open Source Program Office
OTC	Open Source Technology Center
OTT	over the top
OU	Open University
OWR	open when ready
P2P	person-to-person
para(s)	paragraph(s)
PLoS	Public Library of Science
PR	public relations
PUBPAT	Public Patent Foundation
QAA	Quality Assurance Agency
R&D	research and development
RAND	reasonable and non-discriminatory terms
RCP	Rich Client Platform
RDFa	Resource Description Framework in Attributes
RF	royalty free
RHEL	Red Hat Enterprise Linux
RIT	Rochester Institute of Technology
RMS	Richard M Stallman
ROI	return on investment
RPC	remote procedure call
RPM	RPM Package Manager
SaaS	Software as a Service
SBOM	software bill of materials
SCO	SCO Group
SDO	Standards Development Organization
SEC	Securities and Exchange Commission
SEP	Standard Essential Patent
SME	small and medium-size enterprise
SOW	Scope of Work

SPDX	Software Project Data Exchange
SSPL	Server-Side Public Licence
SV	Satoshi's Vision
TCO	total cost of ownership
TDF	The Document Foundation
TEU	Treaty on European Union
TFEU	Treaty on the Functioning of the European Union
TPM	technological protection measures
TRIPS	Trade-Related Aspects of Intellectual Property Rights
UK	United Kingdom
UN	United Nations
UNESCO	United Nations Educational, Scientific and Cultural Organization
UPC	Unified Patent Court
UPC	Unique Production Code
US	United States
USC	United States Supreme Court
USPTO	US Patent Office
VC	venture capital
W3C	World Wide Web Consortium
WIPO	World Intellectual Property Organization
WTO	World Trade Organization

Contributors

Amanda Brock is CEO of OpenUK, the UK organisation for the business of Open Technology (open source software, open hardware and open data); elected Board Member, Open Source Initiative; appointed member of the Cabinet Office's Open Standards Board; Member of the British Computer Society Inaugural Influence Board; Advisory Board Member, KDE, Planet Crust, Sustainable Digital Infrastructure Alliance and Mimoto; Charity Trustee Creative Crieff and GeekZone; and European Representative of the Open Invention Network. Amanda was awarded the UK Lifetime Achievement Award in the Women, Influence & Power in Law Awards, 2022, and included in Computer Weekly's Most Influential Women and The UK Leaders in Tech long lists in 2021 and 2022. A lawyer of 25 years' experience, she previously chaired the Open Source and IP Advisory Group of the United Nations Technology Innovation Labs, sat on the OASIS Open Projects and UK Government Energy Sector Digitalisation Task Force Advisory Boards and been an advisor to a number of start-ups including Beamery and Everseen. With law degrees from the University of Glasgow, New York University and Queen Mary and Westfield, University of London, Amanda was part of the first cohort to study internet law in the UK. She then spent 25 years practising law and almost 20 of those across companies in a variety of sectors, with a strong technology focus. She was the first lawyer working on the ISP Freeserve from 1999 and a member of the team which took it to IPO. She joined Canonical early stage as General Counsel setting up and running the global legal team for 5 years from 2008. A frequent international keynote speaker, Amanda writes regularly for the technology press, is Editor of Open Source, Law, Policy and Practice, being published by Oxford University Press in October 2022 with open access sponsored by the Vietsch Foundation. Listed as one of 20 CEO's to Watch at https://www.linkedin.com/feed/update/urn:li:activity:6777656310428135424/, https://www.linkedin.com/in/amandabrocktech

Malcolm Bain is an English solicitor and Spanish abogado based in Barcelona, working for the last twenty years in ICT law and focusing on Open Source and open content projects. He has advised universities, government, industry, and startups on intellectual property strategy, management, and licensing, and participated in many conferences and seminars on the legal aspects of Open Source and open data. He is a member of FSF-Europe and CATPL, the Catalan association for free software businesses.

Miriam Ballhausen is a partner at Bird & Bird, LLP, specialising in technology, software, digital media, copyright, data, and data protection law with a particular focus on the collaborative development of Open Source software, open data, and open hardware. She served on the advisory council of the Legal Network of the Free Software Foundation Europe and has been involved in several Open Source enforcement cases in Germany. In her daily work, she regularly works with clients on implementing Open Source licence compliance program and advises them on all issues related to Open Source.

Knut Blind took his Bachelor's degree at Brock University (Canada), prior to finishing his Diploma in Economics and later his doctoral degree at Freiburg University. Since 1996, he joined the Fraunhofer Society. Currently, he is head of 'Innovation and Regulation' at the Fraunhofer Institute for Systems and Innovation Research located in in Karlsruhe, Germany. In April 2006, he was appointed Professor of Innovation Economics at the Faculty of Economics and Management at the Technische Universität Berlin. Between 2008 and 2016, he held also the endowed chair of standardisation at the Rotterdam School of Management of the Erasmus University.

Mirko Böhm is an Open Source software contributor to the KDE Desktop, the Open Invention Network, the Open Source Initiative, and other projects. He is a visiting lecturer and researcher on Open Source software at the Technical University of Berlin, a certified Qt specialist and trainer and a fellow of the Openforum Academy. He leads software engineering projects at Mercedes-Benz where he applies a wide range of experience as an entrepreneur, corporate manager, software developer, and German Air Force officer. He lives with his wife, two children, and two cats in Berlin, Germany.

Michael Cheng is a former network engineer, M&A Attorney, and product manager. He is currently Vice President, Head of Corporate, Mergers & Acquisitions, and Intellectual Property at Dapper Labs. Prior to this, Michael was a product manager at Facebook/Meta where he represented the company as the face of its investments in Open Source. He has previously served on the Linux Foundation Board of Directors (Member), ML Commons (Treasurer), Confidential Computing Consortium (Board Member), Urban Computing Foundation (Board Member), OpenChain (Board Member), Open Invention Network Technical Advisory Committee (Member), and the Magma Foundation (Board Chair).

Pamela S. Chestek is the principal of Chestek Legal in Raleigh, North Carolina. She counsels creative communities on Open Source software, brand, and marketing matters. Prior to returning to private practice, she held in-house positions at footwear, apparel, and high technology companies and was an adjunct law professor teaching a course on trademark law and unfair competition. She is a frequent author of scholarly articles, and her blog, *Property, Intangible*, provides analysis of current intellectual property case law. She is admitted to practice in California, Connecticut, the District of Columbia, Massachusetts, New York, and North Carolina, and has been certified by the North Carolina Board of Legal Specialization in Trademark Law.

Mishi Choudhary is a technology lawyer. The *Open* magazine calls her an emerging legal guardian of the free and open Internet. She is the Legal Director of the New York-based Software Freedom Law Center and Partner at Moglen & Associates. She has served as the primary legal representative of many of the world's most significant free software developers and non-profit distributors. She advises technology startups and established businesses around the world on Open Source software licensing and strategy. In 2010, she founded SFLC.in. Under her direction, SFLC.in has become the premier non-profit organisation representing the rights of Internet users and free software developers in India.

Shane Coughlan is an expert in communication, security, and business development. His professional accomplishments include building the largest Open Source governance community in the world through the OpenChain Project, spearheading the licensing team that elevated Open Invention Network into the largest patent non-aggression community in history and establishing the first global network for Open Source legal experts. He is a founder of both the first law journal and the first law book dedicated to Open Source. He currently leads the OpenChain Project, acts as an advisor to both World Mobile and Asylum Labs, and is a General Assembly Member of OpenForum Europe.

Toby Crick is a partner in Bristows LLP's technology group and advises on and negotiates commercial, technology, and outsourcing agreements. He has particular expertise in working with clients to help them manage and structure complex deals and is recognised for his work on digital transformation projects and his work with clients to manage Open Source software in regulated environments. He is a Trustee of the UK's Society for Computers and Law and lectures widely on IT, e-commerce, cloud computing, agile software development, and outsourcing including at ITechLaw, University College London (where he teaches on Open Source) and Queen Mary University of London.

Richard Fontana is Senior Commercial Counsel at Red Hat. He specialises in legal matters relating to software development, with a focus on Open Source. He is a former board director of the Open Source Initiative. Fontana was previously Senior Director and Associate General Counsel for Cloud and Open Source at Hewlett-Packard and Counsel at the Software Freedom Law Center. Earlier in his career he practised intellectual property and antitrust law. He is a graduate of the University of Michigan Law School (Juris Doctor), Yale University (Master of Science in Computer Science), and Wesleyan University (Bachelor of Arts in History).

Ross Gardler has been working on Open Source software for close to twenty-five years, participating in many projects with a focus on building healthy collaboration environments that create opportunities for open innovation across multiple fields. He served for a number of years as the President of the Apache Software Foundation and currently serves on the Board of Directors for OASIS Open at the intersection of rapid Open Source software innovation and stable interoperability through the slower but more precise standards process. He currently works for Microsoft contributing to the growth of Linux workloads on Azure.

Andrew Katz is a solicitor practising in England and has specialised in open technologies for over 25 years. He leads the Technology team at Moorcrofts LLP in the Thames Valley and has advised businesses, governments, non-governmental organisations, and foundations around the world on open licensing and governance. He is co-author of the CERN Open Hardware Licence, and is a visiting researcher at Queen Mary University of London and the University of Skövde. He lectures frequently, and has written numerous papers on open technologies. He was lead open hardware author on the European Commission's 2021 Paper on the Impact of Open Source Software and Hardware on the EU Economy. He has written and released software under the GPL.

Jilayne Lovejoy is a US lawyer and community leader who has held various community and in-house roles related to Open Source. She co-leads the Software Package Data Exchange® (SPDX) legal team, helps maintain the SPDX License List, and co-authored the FINOS Open Source License Compliance Handbook, an open-licensed human and machine-readable handbook for licence compliance. Currently, she is product counsel at Red Hat working on a variety of issues. Prior roles include legal counsel at Canonical and principal Open Source counsel at Arm, where she drove improved processes related to Open Source, including forming and chairing the Arm Open Source Office.

P McCoy Smith is the Founding Attorney at Lex Pan Law LLC, a full-service technology and intellectual property law firm, and Opsequio LLC, an Open Source consultancy, both in Portland, Oregon, USA. He spent 20 years at a Fortune 50 multinational technology company as an intellectual property attorney, where he ran Open Source legal policies. He spent eight years in private practice, as a patent litigator and prosecutor, in a New York City-based law firm, and three years as a patent examiner at the US Patent & Trademark Office. He has an honours engineering degree (Colorado State University), a graduate liberal arts degree (Johns Hopkins University), and a law degree (University of Virginia). He also taught the US patent bar exam, and is on the editorial board of the *Journal of Open Law, Technology & Society*. He is licensed to practice law in Oregon, California, and New York, and to prosecute patent and trademark applications in the US and Canadian Patent & Trademark Offices.

Iain G Mitchell KC is a member of the Scottish and English Bars, ranked in *Chambers Directory* and the *Legal 500* for Commercial Litigation, Intellectual Property and Information Technology law. He is Chair of the Scottish Society for Computers and Law, the UK expert on the IT Committee of the CCBE, and past Chair of its Surveillance Working Group. He is a member of the IT Panel of the Bar Council of England & Wales. the legal panel of Open UK and an Honorary Lecturer in IT Law at Münster University. He is a joint editor of the *Journal of Open Law, Technology and Society.*

Cristian Parrino is a tech turned social entrepreneur and sustainability advisor. He is OpenUK's Chief Sustainability Officer where he focuses on the intersection of open technology and societal value. He is also the CEO of childhood cancer charity, the Azaylia Foundation, and a Board Trustee at citizen science charity, Earthwatch Europe, and youth climate action charity, InterClimate Network, where he also co-leads on the Youth Action Against Climate Change All-Party Parliamentary Group. Previously, he was the co-founder and CEO of sustainable behaviour change startup Greengame, and the Vice President of Mobile and Online Services at Canonical.

Carlo Piana is a qualified lawyer in Italy and an Open Source software advocate. Former General Counsel to the Free Software Foundation Europe, which he represented along with the Samba Team in cornerstone antitrust cases to ensure freedom of interoperability in the PC and Internet market. In the 2022 he was elected to the Board of the Open Source Initiative and became a member of the Steering Committee of the Eclipse Oniro Working Group. He acted in the first reported GPL enforcement case in Italy. He is a founding member of Array, a boutique IT law firm, and a partner of OpenChain.

Mark Radcliffe is a senior partner who practises in DLA Piper's Silicon Valley office and is Co-Chair of its Blockchain and Digital Assets practice. He has been advising on Open Source matters for over twenty years, with projects ranging from the development of the dual licensing model to the open sourcing of the Sun Microsystems' Solaris operating system. He serves as outside general counsel of the Open Source Initiative and Apache Software Foundation on a *pro bono* basis. He is applying this experience to blockchain and non-fungible token issues. He has written and spoken extensively on Open Source legal issues.

Nithya A Ruff is the Head of the Amazon Open Source Program Office. She drives Open Source culture and coordination inside of Amazon and engagement with external communities. Prior to Amazon, she started and grew Comcast and Western Digital's Open Source Program Offices. Nithya has been director-at-large on the Linux Foundation Board for the past five years and in 2019 was elected Chair of the influential Linux Foundation Board. She works actively to advance the mission of the Linux Foundation around building sustainable ecosystems that are built on open collaboration. She is a passionate advocate and a speaker for opening doors to new and diverse people in technology and often speaks and writes on this topic. She graduated with an MS in Computer Science from NDSU and an MBA from the University of Rochester, Simon Business School and is an aspiring corporate board director and governance enthusiast.

Karen Sandler is an attorney and executive director of the Software Freedom Conservancy (SFC), a non-profit organisation focused on ethical technology. She is known as a 'cyborg lawyer' for her advocacy for software freedom as a life-or-death issue. She co-organises Outreachy, an internship program for those facing under-representation, systemic bias, or discrimination in tech. She is a Lecturer-in-Law at Columbia Law School. Prior to SFC, Karen was executive director of the GNOME Foundation. Before that, she was general counsel of the Software Freedom Law Center. She received her JD from Columbia Law School where she was a James Kent Scholar, and her engineering degree from the Cooper Union.

Charles-H Schulz is a French technologist, free software and open standards advocate. As a long-time contributor to the OpenOffice.org project, he helped grow its community from a few, mostly European communities to over 100 communities and teams of various sizes. He also contributed to the development and adoption of the OpenDocument Format standard through the company he co-founded, Ars Aperta. A former director of the OASIS Consortium, he has engaged in various digital public policy debates at the European level. He is a founding member and one of the former directors of the Document Foundation, home of the LibreOffice project. He has been working in governmental cybersecurity for several years and is one of the current board members of the Open Information Security Foundation, and Chief Strategy Officer of Vates, the main developer of the XCP-ng hypervisor.

Kate Stewart works with the safety, security, and licence compliance communities to advance the adoption of best practices into embedded Open Source projects. She was one of the founders of SPDX and is currently the specification coordinator. Since joining the Linux Foundation, she has launched the ELISA and Zephyr Projects, as well as supporting other embedded projects. With over thirty years of experience in the software industry, she has held a variety of roles in software development, architecture, and product management, primarily in the tooling and embedded ecosystem working with international teams.

Dr Nikolaus Thumm is Senior Scientific Advisor with the ETH Board in Zurich, Switzerland, and Associate with Technical University Berlin. Prior to this, he worked for the European Commission where he was responsible to set up a work program on standardisation, standard essential patents, licensing, and Open Source. Until 2013, he was Chief Economist of the European Patent Office. Before this, he worked as Senior Economic Counsellor for the Swiss Intellectual Property Office. He was chairman of the United Nations' Advisory Group on the Protection and Implementation of Intellectual Property Rights for Investment, a private-public partnership group. Nikolaus lead numerous international research activities and holds many publications in the field of standardisation, patenting, and intellectual property protection.

Dr Ian Walden is Professor of Information and Communications Law and Director of the Centre for Commercial Law Studies, Queen Mary University of London. His publications include *Media Law and Practice* (2009), *Free and Open Source Software* (2013); *Computer Crimes and Digital Investigations* (2nd edn, 2016) and *Telecommunications Law and Regulation* (5th edn, 2018). He has been a visiting professor at the universities of Texas, Melbourne and KU Leuven. He has been involved in law reform projects for the World Bank, European Commission, Council of Europe, Commonwealth, and UNCTAD, as well as numerous individual states. Ian was an 'expert nationaux détaché' to the European Commission (1995–96); Board Member and Trustee of the Internet Watch Foundation (2004–09); on the Executive Board of the UK Council for Child Internet Safety (2010–12); the Press Complaints Commission (2009–14); a member of the RUSI Independent Surveillance Review (2014–15); a member of the Code Adjudication Panel at the Phone-paid Services Authority (2016–21); a member of the European Commission Expert Group to support the application of the GDPR (2017–21), and a Non-Executive Board Member of the Jersey Competition Regulatory Authority (2020–present). Ian is a solicitor and Counsel to Baker McKenzie. Ian leads Queen Mary's qLegal initiative and is a principal investigator on the Cloud Legal Project.

Stephen Walli is a principal program manager in the Open Source Ecosystem Team in the Azure Office of the CTO. He has collaborated on standards and Open Source projects for more than thirty years. He is a board member to the Eclipse Foundation and chairs the Eclipse SDV Working Group, and chairs the Confidential Computing Consortium (Linux Foundation). He is also adjunct faculty at Johns Hopkins, teaching Open Source Software Engineering, and is developing the Semesters of Code program. He is working group chair for an IEEE standard on recommended practices for Open Source software project governance.

Table of Cases

Parking Brixen GmbH v Gemeinde Brixen & Stadtwerke Brixen AG
(Case C-458/03) [2005] ECR 1–08585. . . . . .433, 439
Poucet and Pistre (Joined Cases C-159/91 and C-160/91) [1993] ECR I-637. 436

SAS Institute Inc. v World Programming Ltd (Case C-406/10)
ECLI:EU:C:2012:259, 2 May 2012 . . . . 4, 76, 229
Sociedad General de Autores y Editores de Espana (SGAE) v Rafael Hotels SL
(Case C-306/05) 7 December 2006; [2007] ECDR 2 28–29
Solvay SA v Honeywell Fluorine Products Europe BV, Case C-616/10
(ECJ 12 July 2012). 224–25
Synergestic herbicides/CIBA GEIGY T68/85 254

Telaustria Verlags GmbH and Telefonadress GmbH v Telekom Austria AG
(Case C-234/98) [2000] ECR 1–10770. . . . . 433, 438–39

UsedSoft GmbH v Oracle International Corp (2012) 3 CMLR 44 27

EUROPEAN COURT OF JUSTICE (*CHRONOLOGICAL*)

Case 85/76 Hoffman-La Roche, judgment of the European Court of Justice
(Hoffmann-La Roche & Co. AG v Commission of the European
Communities. Dominant position. 370–71
Case T-68/85 Synergestic herbicides/CIBA GEIGY. 254
Cases C-159/91 and C-160/91 Poucet and Pistre [1993] ECR I-637 436
Case C-359/93 European Commission v The Netherlands [1995] ECR I-15 440
Case C-67/96 Albany [1999] ECRI-5751. 436
Case C-200/96 Metronome Musik [1998] ECR I-1953 26
Case C-234/98 Telaustria Verlags GmbH and Telefonadress GmbH v
Telekom Austria AG [2000] ECR 1–10770 . . . . 433, 438–39
Case C-496/99 Commission v CAS Succhi di Frutta SpA [2004] ECR 2004 I-03801. 445
Case C-513/99 Concordia Bus Finland Oy Ab [2002] ECR I-7251 441
Case C-59/00 Bent Mousten Vestergaard v Spøttrup Boligselskab
[2001] ECR 1–09505 . . . . .433, 440
Case C-448/01 EVN AG and Weinstrom GmbH v Austria [2003] ECR I-14558 441–42
Cases C-21/03 and C-34/03 Fabricom SA v Belgium [2005] ECR I-1577 439
Case C-205/03 Nacional de Empresas de Instrumentación Científica, Médica,
Técnica y Dental (FENIN) v Commission of the European Communities
[2006] 5 CMLR 7 436
Case C-458/03 Parking Brixen GmbH v Gemeinde Brixen & Stadtwerke
Brixen AG [2005] ECR 1–08585. . . . . .433, 439
Case T-201/04 Microsoft v Commission 374–76
Case C-479/04 Laserdisken ApS v Kulturministeriet [2007] 1 CMLR 6 26
Case C-6/05 Medipac-Kazantzidis AE v Venizeleio-Pananeio [2007] ECR 1–04557 433
Case C-306/05 Sociedad General de Autores y Editores de Espana (SGAE) v
Rafael Hotels SL 7 December 2006; [2007] ECDR 2. 28–29
Case T-319/9 Nacional de Empresas de Instrumentación Científica, Médica,
Técnica y Dental (FENIN) v Commission of the European Communities
[2003] 5 CMLR 1 436
Case C-393/09 Bezpečnostní softwarová asociace [2010] ECR I-0000 4
L'Oréal SA & Ots v Bellure NV & Ots [2010] ECJ C-487/07 196
Case T-292/10 Monty Program AB v Commission 382
Case C-406/10 SAS Institute Inc. v World Programming Ltd 2 May 2012 . . . . 4, 76, 229

Case C-616/10 Solvay SA v Honeywell Fluorine Products Europe BV (ECJ 12 July 2012). . . . 224–25
UsedSoft GmbH v Oracle International Corp (2012) 3 CMLR 44 27
Case C-607/11 ITV Broadcasting Ltd & ors v TVCatchup Ltd 7 March 2013. . . . 28–29
IT Development SAS v Free Mobile SAS, ECLI:EU:C:2019:1099 (Fifth Chamber, CJEU, 18 December 2019) 105–6

FRANCE

EDU 4 v AFPA, Cour d'Appel de Paris, Pole 5, Chambre 10, no: 294 104–5

Entre'Ouvert v Orange, Tribunal de grande instance (TGI) of Paris, 3rd chamber, 3rd section (21 June 2019). . . . 105–6

GERMANY

Anmerkung zu OLG Düsseldorf, U. v. 24.04.2012–I-20 U 176/11 186

Welte v D-Link Deutschland GmbH (2006) LG Frankfurt a.M., 2006–09–06, Case No. 2–6 O 224/06. . . . 29
Welte v Sitecom Deutschland GmbH, District Court of Munich, 19 May 2004, case 21 O 6123/04. . . . 104–5
Welte v Skype Technologies S A, District Court of Munich, 12 July 2007, case 7 O 5245/07. . . . 104–5

ITALY

Assoli v Ministero del Lavoro (TAR (Regional administrative court) Lazio, Decision no 428 of 23/01/2007 447

Consorzio Aziende Metano (Coname) v Comune di Cingia de'Botti C-231/03 [2005] ECR I-07287 433

Decision no. 122 of 22/03/2010. . . . 447

SECAP SpA and Santorso Soc. Cooparl v Comune di Torino C-147/06 and C-148/06 [2008] ECR I-03565 433

POLAND

Decision of Krajowa Izba Odwoławcza of 13 January 2009 (file no: KIO/UZP 1502/08) 449
Decision of Krajowa Izba Odwoławcza of 5 August 2009 (file no: KIO/UZP 961/09) 449

Judgment of Constitutional Court of 9 March 1988, file no: U 7/87, OTK 1988, no 1, poz 1, 14 448–49

UNITED KINGDOM

Aerotel Ltd v Telco Holdings Ltd [2007] RPC 7 218

Boosey v Whight, 1900 1 Ch. 122, 81 LTNS 265 72–73

Carlill v Carbolic Smoke Ball Company [1892] EWCA Civ 1 504–5
Confetti Records v Warner Music UK Ltd (t/a East West Records) [2003] EWHC 1274 12–13

Elekta Ltd v Common Services Agency 2011 SLT 815 . . . . .442, 448

Football Association Premier League Ltd and others v QC Leisure and others, Murphy v Media Protection Services Ltd [2012] 1 CMLR 29 26

Halliburton Energy Inc's Patent [2011] EWHC 2508 (Pat) 218

Macrossan's Application 2006 [EWCA] 218

Navitaire Inc. v Easyjet Airline Co [2004] EWHC 1725 (Ch) 75

R (on the application of Chandler) v Secretary of State for Children, Schools and Families (2010) CMLR 19 434

SAS Inst. Inc. v World Programming Ltd. [2013] EWCA Civ 1482 79
SAS Institute Inc. v World Programming Ltd [2013] EWHC 69 (Ch) 4
Sidey Ltd v Clackmannanshire Council and Pyramid Joinery and Construction Ltd [2010] SLT 607 434
Sidey Ltd v Clackmannanshire Council [2012] SLT 334 . . . . .434, 438
St Albans City & DC v International Computers Ltd., [1996] 4 All ER 481 6
Symbian Ltd v Comptroller General of Patents [2008] EWCA Civ 1066 218

The Mayor and Burgesses of the London Borough of Southwark v IBM UK Ltd [2011] EWHC 549 (TCC) 6

UNITED STATES

100 Blacks in Law Enforcement Who Care, Inc. v 100 Blacks Who Care, Inc., Opposition No 91190175, 2011 WL 1576733 (TTAB 12 April 2011) 202

American Axle & Manufacturing, Inc. v Neapco Holdings LLC, 939 F.3d 1355 (Fed. Cir. 2019) 220
Apple Computer, Inc. v Franklin Computer Corp., 714 F.2d 1240 (3d Cir. 1983), cert. dism'd by stip., 464 US 1033 (1984) 72–73
Apple Inc. v Psystar Corp., 658 F.3d 1150, C.A.9 (Cal.), 2011 26
Autodesk, Inc v Dassault Systemes SolidWorks Corp, 685 F. Supp. 2d 1001, 1009 (N.D. Cal. 2009). 209

Baker v Selden, 101 US 99 (1879) 76
Barcamerica Int'l USA Tr v Tyfield Importers, Inc, 289 F.3d 589, 598 (9th Cir. 2002) 196
Bedrock Computer Technologies LLC v Softlayers Technology Inc., Case No. 6:09–cv-269 (LED) (E.D. Tex. 2009) 249–50
Bedrock Computer Technologies LLC v Softlayers Technology Inc., Case No. 6:09–cv-269 (LED) (E.D. Tex. 15 April 2011). 249–50
Beltronics USA Inc v Midwest Inventory Distrib, LLC, 562 F.3d 1067, 1072 (10th Cir. 2009) 199, 200
Bernstein v US Department of State 176 F.3d 1132 (9th Cir. 1999) 10
Bilski, In re, 545 F.3d 943 (Fed Cir 2008) (en banc) 219–20
Bilski v Kappos, No 08–964, 561 U.S. (2010). 219–20

Table of Legislation

INTERNATIONAL INSTRUMENTS

(alphabetical list)

Introduction

Amanda Brock, Editor

> 'Never doubt that a small group of thoughtful, committed citizens can change the world: indeed, it's the only thing that ever has.'
>
> American Cultural Anthropologist, Margaret Mead

It has been a true privilege to be part of a small group of thoughtful, committed citizens (which grew to be a big group) over the fourteen years since I first stumbled into Open Source by joining Canonical, in February 2008. Open Source is undoubtedly changing the world through collaboration, and the community generated from this collaboration is special to me.

I use the word community a lot. If you wonder what I mean by it, I am referring to the ecosystem of people working and contributing to Open Source software codebases and their environments. Tolerance and understanding sit at the heart of community. It is certainly at the heart of our Open Source legal and policy community and most definitely my Open Source community, a group of people, ever increasing, who have become an extended family to me, working collaboratively to ensure the sustainability of our technology ecosystem.

Most of those I asked to contribute agreed to write for this book. None of them were paid. They have deep expertise and I am humbled to edit their work. I am extraordinarily grateful.

None of them are *your* lawyers and none are giving *you* legal advice. They have shared the benefit of years of experience and hard work. I hope that you will find this book helpful, but it cannot be a substitute for you taking legal advice when you need it. Also, each author has written their individual contributions and none have reviewed or endorsed the others'. End of disclaimer ;-). Please use the book freely and I hope it helps you to collaborate and build great things.

The first edition, edited by Noam Shentov and Ian Walden, was an inspiration and has allowed us to expand this edition into a more global text. Their work at Queen Mary, my alma mater, is greatly appreciated. Thanks to Oxford University Press for recognising the need for this publication in its first edition.

The excitement of having the Vietsch Foundation fund this book being open access is hard for me to express. I could not be more grateful to them and to NLNet's Michiel Leenars, for helping me to find that funding. Thank you both. This book being open access is a gift to anyone who wishes to teach Open Source. Herein lies your curriculum for the non-coding aspects of Open Source. There is no better source today. There is no similar text. It is also a gift to communities and businesses on their journey through Open Source maturation and learning how to curate their Open Source software.

This book is a one-off. It would take decades to pull together the experience to create a similar work. I will *not* edit a third version alone, as I could never have foreseen the scale of work involved, and any future versions will be edited with co-editors only.

Thank you to all of the Open Source projects who have allowed us to use their logos for the cover. I love that we were able to make this happen.

1

Open Source as Philosophy, Methodology, and Commerce

Using Law with Attitude

Ian Walden

1.1 Introduction

Software (in my opinion) is the really clever bit of computing! Software is much, if not most, of what we know as the information and communications technology (ICT) industries. Software commonly comprises two written forms: source code and object code.[1] Source code is the language in which computer programs are generally written and which is then compiled into machine-readable object code for use by the processor, either in a form distinct from the hardware or incorporated into it, i.e. firmware. There are numerous programing languages structured at differing levels of abstraction, representing different generations of programing language. The conversion between source code and object code cannot be easily inverted, and this acts as an effective control over the use made of object code (for interpreted languages, the source code can be obfuscated, meaning that although it still runs, it is more difficult for a human to understand it). As a consequence, 'traditional' commoditised proprietary software has been distributed in object code form, rather than source code, rendering modification of the software difficult.

[1] Although the popularity of so-called interpreted languages such as Python which is executed directly from the source is increasing significantly.

Ian Walden, *Open Source as Philosophy, Methodology, and Commerce* In: *Open Source Law, Policy and Practice*.
Edited by: Amanda Brock, Oxford University Press. © Ian Walden 2022. DOI: 10.1093/oso/9780198862345.003.0001

The free and open source software movements (collectively referred to as 'Open Source') subverts this traditional industry model by providing access to the original source code for the user. Such access enables further development of the software, amending the existing code or writing new lines of code, for personal or public benefit. The motivations of those that pursue an 'Open Source' approach to software vary considerably, encompassing political, philosophical, and ethical agendas as much as simple pragmatism. While acknowledging and examining this spectrum of motives, the editorial stance of the book is one of attempted neutrality in order to understand and analyse the phenomenon of 'Open Source' as a legal construct.

The central fact of Open Source, the fact that justifies this book, is that maintaining control over source code relies on the existence and efficacy of intellectual property (IP) laws, particularly copyright law. Copyright law is the primary statutory tool that achieves the end of openness, although implemented through private law arrangements at varying points within the software supply chain. This dependent relationship is itself a cause of concern for some philosophically in favour of 'open', with some predicting (or hoping) that the free software movement" will bring about the end of copyright as a means for protecting software.[2]

This book examines various policies, legal, and commercial aspects of the Open Source phenomenon. For our purposes, Open Source is adopted as convenient shorthand for a collection of diverse users and communities, whose differences can be as great as their similarities. The common thread is their reliance on, and use of, law and legal mechanisms to govern the source code they write, use, and distribute.

This chapter has three main objectives. First, to introduce the subject matter, Open Source, the environment, and many of the themes that are examined and analysed throughout this book. Second, the relationship between copyright law and Open Source is scrutinised, mapping areas of common cause and tension, as well as areas of legal uncertainty, from both a theoretical and practical perspective. Finally, the chapter is a study of how private law arrangements can be used to achieve outcomes that diverge from that intended for the applicable public law regime: using law with attitude.

1.2 The Legal Treatment of Software

Before embarking on an analysis of how Open Source is used by software developers and communities, it is necessary to consider the legal treatment of software from a generic perspective, under IP laws and within the wider legal framework.

[2] Eben Moglen, 'Anarchism Triumphant: Free Software and the Death of Copyright' in N Elkin-Koren and NW Netanel (eds), *The Commodification of Information* (Amsterdam: Kluwer Law international, 2002) 107–31.

Open Source proponents utilise private law mechanisms, i.e. licences (the legal nature of a licence can vary, as contracts or bare licences, which is examined further in Chapter 3)[3] and contracts, operating within established public law frameworks such as copyright, patent, and contract law to achieve a particular desired outcome. For our purposes, 'public law' is used differently from the traditional concept concerning the relationship between a state and its people. Here, it encompasses legal regimes that govern relationships between people, both within and between jurisdictions.[4] These regimes not only grant legal validity and enforceability to the private law mechanisms but also directly intervene to influence the use of these mechanisms, prohibiting certain practices, for example, and making determinations about certain conduct.

Uncertainties about how the public law will treat certain industry practices, including those of the Open Source community, are highlighted throughout the book. They include whether software should be treated as a good or a service, what constitutes a modification, whether usage is governed by contract or bare licence and whether that mechanism results is a transfer of ownership or a right to use. These questions are sometimes answered through legislative provision or judicial interpretation, but rarely without generating further areas of doubt. A private law instrument may itself be expressed in language that can deliberately or accidentally include terms that go beyond what is recognised or acceptable in public law but which reside unchallenged and unenforceable, or terms which are interpreted differently by different people or groups through inference or philosophical bent.[5] Collectively, such legal uncertainties can have a negative impact on technical and commercial innovation and development in the ICT industries. As such, one aim of the book is to try and address some of the uncertainties that surround Open Source.

In the early days of computing, software was distributed free with hardware, becoming a commodity only when it became liberated from the hardware on which it operated.[6] With the emergence of software as a discrete item, an issue arose as to the most appropriate regime within IP law under which to protect it. The three leading possibilities were patent law, due to its industrial nature; copyright law, as a form of expression, or the establishment of some *sui generis* regime that reflected the unique features of software.[7] The law of confidentiality and trade secrets were

[3] This chapter uses this term in a non-specific manner.

[4] This includes regulatory and judicial law-making, as well as international and EU law.

[5] See, e.g., M Herman and J Montague, 'The elephant in the room: Patent value and FOSS', paper presented at the AIPLA Spring Meeting, San Francisco, CA, April 2011. Available at <https://docplayer.net/8677141-The-elephant-in-the-room-patent-value-and-open-source-software-michele-herman-davis-wright-tremaine-llp-and.html> accessed 21 July 2022.

[6] M Schellekens, 'Free and Open Source Software: An Answer to Commodification?' in L Guibault and B Hugenholtz (eds), *The Future of the Public Domain: Identifying the Commons in Information Law* (Amsterdam: Kluwer Law International, 2006), at 309.

[7] See, e.g., the WIPO 'Model Provisions on the Protection of Computer Software', adopted in 1977.

also considered (as discussed later in this chapter). Copyright eventually won the argument, with computer programs being accepted as a form of 'literary work',[8] although with some jurisdictions adopting a sub-set of *sui generis* rules to reflect some of the unique issues raised by software.[9]

What comprises 'software' or 'computer programs', however, often remains less clear.[10] As noted earlier, software is generally expressed in two forms, source code and object code, the latter being a 'translation' of the former, but both being protectable subject matter. As with other areas of law, some jurisdictions attempt to define the concept in law,[11] some extend it beyond the source and object code,[12] while others are content to leave it for the courts to interpret on the basis of standard usage.[13] The European Court of Justice (ECJ) examined the scope of the term in *SAS Institute Inc. v World Programming Ltd.*,[14] holding that a 'computer program' does not extend to the functionality of a program, the programing language, or the format of data files, although the latter two may be copyrightable works in their own right (paras 29–46). In not protecting the functionality of a program, the law is constraining the scope of copyright law, which is supportive of an 'open' approach to the treatment of software as a tool. While the court distinguishes between a program and the language in which it is written, the latter often now come in the form of a program and numerous Open Source programing languages have been developed, such as Python and Ruby, themselves licensed under Open Source licences.[15]

As the software industry developed and while uncertainties continued to exist, software developers tended to rely on trade secrets law and contract as the preferred mechanisms for protecting their investment. With the clarification and strengthening of the copyright regime, from the mid-1980s until recently, the software industry has relied on copyright law and licences as the primary means for governing the use of their software assets. Limitations within copyright law, however, have seen people look to patent law as offering an alternative strategy for protecting and exploiting their software (see further Chapters 10). These legal mechanisms have been supplemented by technical controls that enable rights holders to further control the use of their work.

[8] WIPO Copyright Treaty, art 4. See also the Commission Green Paper, *Copyright and the Challenge of Technology*, COM(88) 172 final.

[9] For example, Council Directive 91/250/EEC of 14 May 1991 'on the legal protection of computer programs' (OJ L122/42, 17.5.1991), codified in 2009, as Directive 09/24/EC (OJ L111/16, 5.5.2009), herein referred to as the 'Software Directive'.

[10] In this chapter, 'software' and 'computer programs' or 'programs' are used interchangeably.

[11] For example, US law, at 17 USC § 101.

[12] For example, Software Directive, at art 1(1), includes 'their preparatory design material'.

[13] For example, Copyright Designs and Patents Act 1988 (CDPA).

[14] Case C-406/10, 2 May 2012. See also Case C-393/09 *Bezpečnostní softwarová asociace* [2010] ECR I-0000, at paras 34–41. See the application of the European Court of Justice (ECJ) decision in *SAS Institute Inc. v World Programming Ltd* [2013] EWHC 69 (Ch).

[15] Note that compilers, which convert source code into object code, are also program(s) with Open Source versions, such as Open64.

Legal uncertainties continue to exist for the software industry from the operation of the copyright regime: some are general to copyright law, such as the scope of usage exceptions; some from the application of the general rules to the specifics of software, such as its treatment as a collection, compilation, or database; while others arise from the *sui generis* rules, such as the right to decompile and the applicability of copyright to application programing interfaces (APIs).[16] What comprises copying in a software environment has generated challenges for copyright law particularly in respect of 'non-literal' copying. Code may be written using different programing languages, enabling a person effectively to copy the internal 'structure, sequence and organisation' of another work and/or its external 'look and feel',[17] without copying the form of expression. In some cases, such practices have been held to constitute infringement, while in others, the courts have held that a merger between idea and expression has taken place, rendering the subject matter unprotectable.[18] Current industry developments may also result in a shift away from copyright and patent law and back towards reliance on contract and trade secrets. Cloud computing and Software as a Service (SaaS) becoming the norm has enabled applications to be accessed via networks, meaning that suppliers no longer need to give users either the source code or object code for the programs they use (see further Chapter 9).

The debate also continues to ebb and flow with regard to the patentability of computer programs, with the US and Europe exhibiting differing attitudes towards the issue (see further Chapter 5 at section 5.1). As most Open Source licences emanate from the US, which has a liberal approach to software patenting, one consequence is that these licences have increasingly had to devote space to addressing patent rights, to ensure that the 'open' objectives continue to be maintained. Many within the Open Source community exhibit a greater dislike towards patents, than towards other forms of IP. There have been public campaigns seeking to prevent or end software patenting, for which parallels do not exist concerning copyright.[19] As such, contentious debates in some areas relating to Open Source, such as standards, often appear motivated primarily by an objection to patents rather than copyright (see further Chapter 11).

Another area of controversy within copyright has been the role of technology itself as a mechanism for controlling the use and abuse of software. Technological

[16] *Oracle America, Inc. v. Google, Inc.*, 872 F.Supp.2d 974 (N.D. Cal. 2012), rev'd and remanded, 750 F.3d 1339 (Fed. Cir. 2014).

[17] C Millard, 'Copyright in Information Technology and Data' in C Reed (ed), *Computer Law*, 7th edn (Oxford: Oxford University Press, 2011), at 7.5.

[18] Millard, 'Copyright in Information Technology and Data', see note 17. See also *Oracle America, Inc. v Google Inc.* (2012) 872 F.Supp.2d 974.

[19] Note also that the Pirate Party has campaigned for significant curtailment of the term of copyright, however, in the case of all material, not just software. This has caused Richard Stallman (founder of the Free Software Foundation) to argue against this policy as it relates to software, for which he is in favour of either maintaining longer terms for free software, or establishing a sui generis right, in order to ensure that GPL-style copyleft continues to work.

protection measures (TPM), from 'dongles' to bit-encryption, together with Digital Rights Management (DRM) techniques, emerged with the growth of the software industry as a potentially powerful tool in the armoury of rights holders trying to stem burgeoning, industrial-scale infringement. In the 1996 World Intellectual Property Organization (WIPO) Treaty, such techniques were granted legal recognition and protection under international copyright law,[20] appearing in national laws often in the form of criminal prohibitions.[21] Proponents of Open Source, particularly within the free software movement, have been highly critical of TPM/DRM technologies and their use to constraint end-users, especially where the controls extend beyond that granted to rightsholders under copyright law.[22] In Europe, such concerns were successfully raised with policy-makers, who proceeded to place TPM under legal controls designed to limit their abuse.[23] While the provisions have been heavily criticised for being narrowly drawn, overly complex to apply, and favourable to rights holders, they do represent some form of victory for Open Source proponents.

Distinct from the governance of software through IP laws, software is developed, supplied to users, bought and sold as an asset, and comprises part of almost all modern commercial activity. These activities are generally governed through contractual agreement between the various parties, whether business, consumers, or public administrations, either distinct from, or incorporating, any IP licence terms. While such agreements are primarily established by one or other party or negotiated, certain mandatory rules of national law will shape these agreements and, indeed, generate their own uncertainties. As with the initial doubt over which IP regime should apply to software, there has been an ongoing debate, in both Europe and the US, about how software supply contracts should be characterised, as a sale of goods or services or both, or some *sui generis* category, and the implications this determination has for the rights and remedies of the user and the obligations of the supplier.[24] European consumer protection rules, for example, impose an obligation to supply any available information about interoperability between software and 'digital content', which includes computer programs.[25]

[20] Arts 11 and 12 respectively.

[21] For example, in the US, 17 USC § 1204 and in the UK, CDPA, s 296ZB.

[22] For example, <http://www.defectivebydesign.org/> accessed 21 July 2022.

[23] For example, Directive 01/29/EC 'on the harmonisation of certain aspects of copyright and related rights in the information society' (OJ L167/10, 22.6.2001), herein referred to as the 'Information Society Directive', at art 6(4). While the Information Society Directive, at art 1(2)(a), does not amend or affect the provisions under the Software Directive, the TPM provisions would appear to be applicable to software.

[24] For the UK, see, e.g., *St Albans City & DC v International Computers Ltd*., [1996] 4 All ER 481 and *The Mayor and Burgesses of the London Borough of Southwark v IBM UK Ltd* [2011] EWHC 549 (TCC). For the US, see *Wofford v Apple Inc.* (2011)(Case No 11-CV-0034 AJB NLS—unreported), where the judge held that software was not a tangible good or service for the purposes of California's Consumers Legal Remedies Act. See generally the American Law Institute's *Principles of the Law of Software Contracts*, 2009.

[25] Directive 11/83/EU on consumer rights (OJ L304/64, 22.11.2011), at recital 19 and arts 5(1)(h) and 6(1)(s).

1.3 Open Source as Philosophy and Politics

While this book examines how public and private law is used by Open Source communities and others, such as governments, it is obviously necessary to begin by understanding why the law is being used in this way: what is the end being sought by the means? Inevitably, the whys fall along a broad spectrum, some pursuing and supporting Open Source on the basis of deeply held philosophical beliefs about how information and knowledge should be treated in society, while others are more pragmatic, viewing Open Source as a means of creating better software or reducing costs for users (see further Chapter 2).

If 'Open Source' were simply a development methodology, it would not engender the types of rhetoric which has been deployed by the proprietary software community[26] and vice versa.[27] Even amongst Open Source proponents, the philosophical underpinnings are viewed as starkly different, Richard Stallman controversially and possibly incorrectly noting: 'Open source is a development methodology; free software is a social movement.'[28] Open Source has also been described as 'a kind of recursive philanthropy'[29] because of the manner in which participant developers devote time and energy writing code that they donate to the project community. Copyleft licences often act in practice to require contributions to be made available to the community that created the original Open Source software.[30] This can be seen as a legal limitation on a developer's ability to depart from such philanthropy, i.e. to change his mind

One recent development in the Open Source field is the issuance of Public Source modules that are deliberately not made subject to any licence.[31] One motive behind such behaviour is a philosophical rejection of the bureaucratic governance structures required to make copyright law support Open Source objectives.

[26] For example, in 2001, Steve Ballmer, CEO of Microsoft, described Linux as 'a cancer that attaches itself in an intellectual property sense to everything it touches'.

[27] For example, Richard Stallman: 'Writing non-free software is not an ethically legitimate activity, so if people who do this run into trouble, that's good!', available at <http://lists.kde.org/?l=kde-licensing&m=89249041326259&w=2> accessed 21 July 2022.

[28] Richard Stallman, 'Why open source misses the point of free software', available at <http://www.gnu.org/philosophy/open-source-misses-the-point.html> accessed 21 July 2022.

[29] George Finney, 'The Evolution of GPLv3 and Contributor Agreements in Open Source Software' (2009) 14 *Journal of Technology Law and Policy* 79–105.

[30] Technically, copyleft licences (such as the GPL family) will ensure that, on distribution of software, its source is made available to the *recipient* under the same licence. The distribution (and hence the right to receive the source) may occur privately. None of the major copyleft licences mandates contributions back to the community (including all of the GPL family, which is partially why GPLv3 is so complex to ensure the right of private distribution is maintained). Nonetheless, any private recipient of GPL code can make the source code (and hence those contributions) available to the community, if they so wish. Further common development practices (such as the use of publicly accessible Git-based instances such as Github and Gitlab also mean that contribution back to the community happens as a matter of course, unless they fork the project).

[31] Simon Phipps, 'GitHub needs to take Open Source seriously', *InfoWorld*, 30 November 2012, available at <http://www.infoworld.com/d/Open Source-software/github-needs-take-Open Source-seriously-208046> accessed 21 July 2022.

As noted by one commentator: 'younger devs today are about POSS—Post open source software. f*** the license and governance'. More likely this reflects an issue with GitHub, the dominant repository, not requiring licences and a lack of understanding. GitHub has recently sought to rectify this shortfalling with a notice making clear that no licence means code is not Open Source.[32]

As discussed later in this chapter with respect to the public domain, copyright law cannot be ignored that easily!

Whilst such views may be representative of only a small minority of the developer community, they may also reflect the entry of the 'born digital' generation into the software industry, many of whom have grown up in an ostensibly copyright-free environment, where everything and anything is available from somewhere.

It is beyond the scope of this chapter to analyse the differing shades of belief and motivation that drive those involved in the free and open source movements. However, given the dependency on IP laws, particularly copyright, the following sections consider some philosophical dimensions of copyright law of relevance to the Open Source community, including in the promotion of freedom of expression, in the protection of the paternity and integrity of works, as well as the relationship with the public domain. The last sub-section shifts from the philosophical to the political and examines how Open Source has become incorporated by governments into public policy initiatives in pursuit of a range of objectives.

1.3.1 Freedom of expression

One of the most quoted slogans of the free software movement is 'free' as in 'free speech', not as in 'free beer'.[33] To achieve this free speech, copyright law is used to facilitate reuse new expression through modification and prevent exclusivity. The term 'copyleft' was chosen to denote that the objective of copyright was being deliberately turned on its head: 'the inverse of "right" '.[34] Copyright is therefore situated as being antithetical to free speech. This perspective is shared by those that view the statutory defences to copyright, such as fair use and fair dealing, as mechanisms for reconciling free speech with copyright.[35]

Yet this has not always been, indeed is not now, the only way of viewing the relationship between copyright and free speech. As noted by the US Supreme Court, as recently as 1985: 'it should not be forgotten that the Framers intended copyright itself to be the engine of free expression. By establishing a marketable right to the use of one's expression, copyright supplies the economic incentive to create

[32] See <https://github.com/readme/guides/open-source-licensing≥ at tl;dr accessed 21 July 2022.

[33] <http://www.gnu.org/philosophy/free-sw.html> accessed 21 July 2022.

[34] 'What is copyleft?', at <http://www.gnu.org/copyleft/> accessed 21 July 2022.

[35] See, e.g., Patrick Masiyakurima, 'The Free Speech Benefits of Fair Dealing Defences' in P Torremans (ed), *Intellectual Property and Human Rights* (Amsterdam: Kluwer Law International, 2008) 235–56.

and disseminate ideas.'[36] Here copyright is seen as being supportive of free speech through the granting of exclusive economic rights. Those rights obviously also exclude certain types of speech, but, so the reasoning goes, as long as the totality of copyright as an 'incentive to create' is greater that the effect of the constraint, the net outcome is beneficial and copyright law can rightly claim to be a tool of free speech.[37] Alternatively, it has been argued that the purpose of copyright should not be seen as a spur to creativity and a societal distributive mechanism but rather as a means to 'affirm the inherent dignity of the author as a speaking being', where acts of infringement are viewed as compelled speech and defences as enabling the communicative acts of others.[38] Whichever perspective you adopt, this 'paradox' between copyright as both an enemy and friend of free speech has been the subject of ongoing debate.[39]

It is also widely accepted that there has been a shift over recent decades in favour of copyright as constraint. With the emergence of information-based economies, copyright has become central to the protection of economic value in intangible information assets. As copyright's economic importance grew, so did calls for the regime to be extended and strengthened. Greater prevalence, coupled with enhanced rights and more effective and dissuasive sanctions, has resulted in numerous examples of copyright being used to chill speech, whether political, artistic, or commercial.[40]

Free speech or freedom of expression is a human right expressly recognised in most legal systems. In some jurisdictions, particularly the US, free speech is accorded pre-eminent status compared with other rights, such as privacy.[41] In Europe, freedom of expression is granted equal status with other rights, including the right of property, which includes IP.[42] As with copyright, freedom of expression is not absolute, it is limited in scope, and is generally weighed in the balance against other protected rights and values.[43]

[36] *Harper & Row Publishers, Inc. v Nation Enterprises* 471 US 539, 558 (1985). While in *Eldred v Ashcroft* (01-618) 537 US 186 (2003), the Supreme Court noted that 'copyright's purpose is to promote the creation and publication of free expression' (at 219).

[37] For a positive view of this trade-off, see RA Cass and KN Hylton, *Laws of Creation* (Cambridge, MA: Harvard University Press, 2013). For a negative perspective, see M Boldrin and DK Levine, *Against Intellectual Property* (Cambridge: Cambridge University Press, 2008).

[38] A Drassinower, 'Copyright Infringement as Compelled Speech' in Lever (ed), *New Frontiers in the Philosophy of Intellectual Property* (Cambridge: Cambridge University Press, 2012) 203–24.

[39] See NW Netanel, *Copyright's Paradox* (Oxford: Oxford University Press, 2008) and J Griffiths and U Suthersanen (eds), *Copyright and Free Speech: Comparative and International Analyses* (Oxford: Oxford University Press, 2005).

[40] Netanel, *Copyright's Paradox*, see note 39, 6.

[41] US Constitution, First Amendment, 'Freedom of Religion, Press and Expression'.

[42] Convention for the Protection of Human Rights and Fundamental Freedoms (1950), art 10 and art 1 of the First Protocol. See also the Charter of Fundamental Rights of the European Union (2007), arts 11 and 17.

[43] See, in particular, *Ashby Donald and others v France*, Appl. Nr. 36769/08, ECtHR (5th Sec.), 10 January 2013, and *Neij and Sunde Kolmisoppi v Sweden*, Appl. Nr. 40397/12, ECtHR (5th Sec.), 19 February 2013.

Source code represents a protected form of expression under both free speech and copyright regimes. While aligned with literary works under copyright law, its treatment as a form of expression under a human rights analysis varies considerably, depending on the specific circumstance. Equating source code with speech has resulted in judicial scrutiny when attempts have been made to constrain the distribution of source code. During the 1990s, governments sought to restrain the export of cryptographic software under export control rules; treating such code as 'dual use', having both civil and military application.[44] Export rules (see Chapter 12) have long existed, but in relation to physical items rather than intangible information. In trying to update these rules for a digital era, they inevitably came into conflict with free expression rights. In *Bernstein v US Department of State*,[45] the US Court of Appeals held that the source code of encryption software was expressive speech for the purposes of the First Amendment and that the existing rules, as a form of prior restraint, violated the protection granted under it. Conversely, in *Universal City Studios, Inc. v Corley*,[46] an injunction prohibiting website owners from posting source code enabling the decryption of movies, or providing links to such code, was considered a permissible constraint on speech.

Where copyright and freedom of expression critically differ as legal regimes, however, is in the role of private law mechanisms in the delineation and enforcement of the respective rights and obligations of the parties. Contract and licence are tools of copyright not of freedom of expression, and it is this feature that renders copyright such a powerful tool, both in the hands of proprietary rights holders and, now, for those promoting and protecting Open Source, the 'commons', and 'free culture'.[47] While private law is generally viewed as forming a lower stratum of any legal system, private law engages persons directly in a manner that the 'higher' levels, from the constitution to statutory provision, often fail to do. People are forced, metaphorically rather than literally, to 'agree' to contractual conditions and a licensee must have notice of the licence terms. Notice and consent are both public law requirements of validity and enforceability for private law arrangements, but they are also methods for obtaining individual engagement with the rights and interests of others, even if it is not always supportive. Indeed, it can be said that it is the private law tools of copyright law which enable the Open Source community to reassert copyright's historic role as an 'engine of free expression'.

[44] For example, Wassenaar Arrangement on export controls for conventional arms and dual-use goods and technologies, 'List of dual-use goods and technologies' (December 2019), available at <https://www.wassenaar.org/> accessed 21 July 2022.

[45] 176 F.3d 1132 (9th Cir. 1999).

[46] 273 F.3d 429 (2nd Cir. 2001).

[47] See James Boyle, *The Public Domain: Enclosing the Commons of the Mind* (New Haven, CT: Yale University Press, 2008) and L Lessig, *Free Culture* (London: Penguin, 2004).

1.3.2 Moral rights

To the extent that support for Open Source is driven by moral and ethical concerns, the moral rights regime within copyright law deserves consideration. As Välimäki has noted: ‘One way to look at open source is to see it promoting the original ideals of authors’ inalienable rights to control the integrity and paternity of their personal creations.’[48] This section considers the affinity between moral rights and an Open Source approach.

While copyright is primarily about economic rights, the Berne Convention also grants authors certain moral rights in respect of their works; commonly referred to as the right of paternity or attribution and the right of integrity:

> Independently of the author’s economic rights, and even after the transfer of the said rights, the author shall have the right to claim authorship of the work and to object to any distortion, mutilation or other modification of, or other derogatory action in relation to, the said work, which would be prejudicial to his honor or reputation.[49]
>
> Moral rights reflect a belief, originating in Continental European countries, that an author of a work has interests in the work that ‘transcend the ordinary motives of commercial gain’.[50] While recognised in the Berne Convention, the treatment of moral rights varies significantly between jurisdictions.[51] Common law countries generally elaborate the least comprehensive regimes, with US copyright law adopting the narrowest statutory conception.[52] By contrast, in civil law countries moral rights are often more extensive than those provided for in Berne.[53]

Moral rights exist independently of the economic rights granted under copyright and are inalienable, generally not capable of being assigned to another,[54] although they can usually be waived.[55] This independent existence enables a divergence to appear between the interests of the creator and the owner of a copyright work. Such divergence has the potential to create problems for governance in an Open Source

[48] M Välimäki, *The Rise of Open Source Licensing* (Helsinki: Turre Publishing, 2005).

[49] Berne Convention for the Protection of Literary and Artistic Works (1886), at art 6*bis*(a). Inserted in 1928. Transposed into UK law by Chapter IV of the Copyright Designs and Patents Act 1988, ss 77–89.

[50] MT Sundara Rajan, ‘Moral Rights in Information Technology: A New Kind of “Personal Right”?’ (2004) 12(1) *International Journal of Law and Information Technology* 32–54.

[51] See Elizabeth Adeney, *The Moral Rights of Authors and Performers: An International and Comparative Analysis* (Oxford: Oxford University Press, 2006).

[52] 17 USC § 106A ‘Rights of certain authors to attribution and integrity’, inserted by the Visual Artists’ Rights Act 1990.

[53] For example, the French Intellectual Property Code recognises a right of disclosure (art L 121-2, ‘*droit de divulgation*’) and a right of display (art L 121-4, ‘*droit de repentir ou de retrait*’).

[54] For example, CDPA 1988, s 94.

[55] CDPA 1988, s 87(2) ‘by instrument in writing signed’, although contract or estoppel may operate in respect of informal waivers (s 87(4)). Waiver is not always permissible, e.g. France.

software project were certain collaborating creators to try to assert their moral rights against the entity owning the copyright and exercising control through an Open Source licence.

The paternity right is bolstered in many copyright systems through the evidential presumption that the named author is the copyright holder;[56] although the right of paternity must be asserted by the author, in other words brought to the attention of others, through some means.[57] Open Source licences are clearly supportive of the paternity right, especially in respect of acts of redistribution, generally requiring that any copyright notices be retained, either in copies of the source code, the original package, or the related documentation.

With respect to modifications, the interrelationship between moral rights and Open Source is more complex. Indeed, it has been argued that Open Source could be seen as sundering the traditional link between the integrity of a work and its author, which historically justified the moral rights doctrine.[58] The integrity right is restricted in scope to modifications and other actions which are 'prejudicial' to the author's honour or reputation. Non-prejudicial modifications, such as a derivative work or an 'adaptation', do not constitute an infringement of the right to integrity, although the right of paternity continues to exist.[59] In common law systems, the evidential burden in an infringement action will generally lie with the claimant (i.e. the author) to demonstrate to the satisfaction of a court that prejudice results from the modification to his work.[60] In civil law systems, the courts are more likely to defer to the subjective view of the claimant author as to the work's derogatory treatment.[61] Derogatory treatment could relate to the content of the work itself, i.e. rewritten code, or the context within which the code is placed, for example incorporation within a disreputable application, such as a virus. The former would rarely give rise to a claim, since rewritten code which is poor quality, potentially damaging the reputation of the original author, is unlikely to be taken up by the community, the collaborative peer-review nature of Open Source communities operating as the control mechanism. While actions based on contextual harm could be constrained by the non-discriminatory rights of use granted with the work, this would also raise the possibility of disproportionate interference in the right of free expression, which is a central element of the Open Source movement. As such, the integrity right is more akin to defamation, which is seen both as an aspect of a person's right to privacy as well as an exception to the right to freedom of

[56] CDPA 1988, s 104.

[57] CDPA 1988, s 78.

[58] Severine Dusollier, 'Open Source and Copyleft: Authorship Reconsidered?' (2002–2003) 26 *Columbia Journal of Law & the Arts* 281–96, at 294.

[59] CDPA 1988, s 77(2).

[60] See, e.g., *Confetti Records v Warner Music UK Ltd (t/a East West Records)* [2003] EWHC 1274, at paras 149–157.

[61] Ian Eagles and Louise Longdin, 'Technological Creativity and Moral Rights: A Comparative Perspective' (2004) 12(2) *International Journal of Law and Information Technology* 209, at 234.

expression,[62] rather than a mechanism to 'govern modifications' akin to the control paradigm of copyright.[63]

The relationship between moral rights and software varies between jurisdictions. Most make no distinction, others tailor the rights with respect to software,[64] while under English law, computer programs are specifically exempt from the moral rights regime.[65] Such an exemption is not manifest in the Berne Convention or other international copyright instruments and has not been followed in other jurisdictions.[66] One suggested reason for exempting computer programs from the moral rights regime is the dependency of 'programers being able to build on pre-existing programs'.[67] Protecting integrity, in particular, is therefore seen as a potential obstacle to technical progress and development. This argument, however, would seem equally applicable to all forms of right that enable control over the use of information. Another reason given is based on the view that moral rights are not appropriate for technological or functional works, as opposed to 'artistic creations' or expressive works.[68] Such an argument would seem to deny the individuality that can be expressed through programing or the existence of a distinct culture that recognises and celebrates 'elegant' programing techniques and solutions.[69] A third argument has been summarised by the European Commission as follows: 'serious doubts exist as to the suitability of their [moral rights] application to works frequently produced collectively, having a technical, industrial, or commercial character and subject to successive modifications'.[70] As well as expressing reservations about technical/functional works, the key feature of concern is the collective nature of the creative process and the extent of modifications to a work that takes place, common characteristics of Open Source communities (although also features of closed and proprietary development systems). How could a right of attribution and integrity operate effectively within such an environment?

On collective attribution, the problem would seem no different in nature from that applicable to 'joint authorship' under copyright law and, indeed, some moral

[62] See H Fenwick and G Phillipson, *Media Freedom under the Human Rights Act* (Oxford: Oxford University Press, 2006), at 1068–70.

[63] G Vetter, 'The Collaborative Integrity of Open Source Software' (2004) 2 *Utah Law Review* 563–700, at 663.

[64] For example, France, IPC, art L 121–7.

[65] France, IPC, ss 79(2)(a) and 81(2).

[66] The Agreement on Trade-Related Aspects of Intellectual Property Rights (TRIPS Agreement) expressly states that the moral rights specified under the Berne Convention do not bestow rights or obligations under TRIPS (at art 9(1)).

[67] Sundara, 'Moral Rights in Information Technology: A New Kind of "Personal Right"?', see note 50, 47. See also Vetter, see note 63, at 565, who states that a right of integrity 'would be counterproductive to the sequential and successive processes used to develop software' (663).

[68] Sundara, 'Moral Rights in Information Technology: A New Kind of "Personal Right"?', see note 50, at 49 and Vetter, see note 63, at 663.

[69] For one description of this culture, see P Himanen, *The Hacker Ethic and the Spirit of the Information Age* (London: Vintage, 2001).

[70] Commission Communication, Green Paper 'on copyright and the challenge of technology', COM(88) 172 final, 7 June 1988.

right provisions already address such issues.[71] Alternatively, while the rights themselves may not be assignable, the right to enforce can be delegated to some other entity,[72] which would create greater certainty for both the community and the users of the code.

On successive modifications, a clear threshold of what constitutes 'derogatory treatment' would be likely to prevent any excessive assertions of a right to integrity. Indeed, similar to the patent retaliation provisions in Open Source licences (see further Chapter 5), any person wishing to assert his right to integrity would first have to ensure that he is not exposed to any similar such claim from any source code which he modified in the course of producing his contribution, which may itself be a significant threshold issue. Alternatively, the concept of integrity could be recast, shifting the locus of protection from individual modifications to the collective output of the community, integrity being infringed where the 'open' nature of the code is undermined through technical or legal means.

The Open Source Initiative (OSI) is the custodian of the Open Source Definition (OSD) which refers to source code integrity in the following terms:

> 4. Integrity of The Author's Source Code
> The license may restrict Source Code from being distributed in modified form *only* if the license allows the distribution of 'patch files' with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.[73]

The rationale refers both to the right of users to transparency about whose code they are using as well as the author's right to protect their reputation. The right of attribution is not directly referenced, although the OSI content is itself licensed under the Creative Commons Attribution licence. However, user transparency can be seen as the flip side of the right of paternity; viewing the right as an obligation and retention of attribution notices is a common characteristic of licences approved by the OSI as meeting the OSD.

Creative Commons licences, which are not designed for code, refer to moral rights, noting that they are not affected by the licence.[74] By contrast, the European Union Public Licence requires the licensor to waive his moral rights, but only 'in order to make effective the licence of the economic rights' provided for under the licence.[75] The Open Database Licence requires the licensor to waive all moral

[71] For example, CDPA 1988, s 88.

[72] Eagles and Longdin, 'Technological Creativity and Moral Rights: A Comparative Perspective', see note 61, at 216.

[73] <http://opensource.org/osd-annotated> accessed 21 July 2022.

[74] <http://creativecommons.org/licenses/by-nc-sa/3.0/> accessed 21 July 2022.

[75] EUPL, v.1.1 (2007), at Clause 2.

rights 'to the fullest extent possible' or agree not to assert such rights. If neither option is permitted by law, the licence cryptically states that 'the author may retain their moral rights over certain aspects of the Database', without specifying what such aspects may be.[76] The GNU General Public License (GPL) makes no reference to moral rights, which reflects its US origins.

It has been argued that moral-type rights should be recast for a digital age, rather than abandoned or avoided.[77] Others have suggested that a distinction could be made between the application of moral rights to object code and source code, especially when the former generates an audiovisual work.[78] Were moral rights to be reinvigorated as a category of IP, what impact would it have on the Open Source community? The answer, as with most legal questions, is it depends! As Ginsburg notes, should moral rights in a digital age 'be achieved by conveying more information about the copy, or by controlling the copy itself?'[79] As noted earlier, the attribution right would seem perfectly aligned with the philosophy of the Open Source movement, subject only to the need to facilitate collective attribution which is generally implemented in the code headers. It is with respect to modifications that our historic conception of moral rights may require recasting to reflect the phenomena of Open Source.

1.3.3 The public domain

The philosophy of the Open Source community is to make source code widely and freely available for use. As such, it begs the question: why not place the source code in the public domain, rather than using the tools of copyright law to achieve the same ends?

The concept of 'public domain' information has a specific meaning within IP law distinct from the state of the information being publicly available.[80] In at least one context, the term 'Open Source' has been used in law as a synonym for publicly available data, rather than software-related data.[81] A 'public domain' work is not subject to any IP rights; it is an alternative state in which information may be. The literature sometimes confuses these two states. Schellekens, for example, notes that software in its pre-commodification state 'belonged to the public domain', which incorrectly equates free, as in speech or beer, with free as in without IP protection.[82] While Boldrin and Levine describes the Open Source movement as having

[76] ODbL v.1.0, at Clause 5.

[77] See, e.g., Jane Ginsburg, 'Have Moral Rights Come of (Digital) Age in the United States' (2001) 9(1) *Cardoza Arts & Entertainment Law Journal* 9–19, 9.

[78] For example, Vetter, see note 63.

[79] Ginsburg, 'Have Moral Rights Come of (Digital) Age In the United States', see note 77, at 17.

[80] See L Guibault and B Hugenholtz, *The Future of the Public Domain: Identifying the Commons in Information Law* (Amsterdam: Kluwer Law International, 2006).

[81] Council of Europe Convention on Cybercrime (2001), at art 32(a).

[82] Schellekens, 'Free and Open Source Software: An Answer to Commodification?', see note 6, at 309.

'relinquished its intellectual monopoly',[83] which implies an abandonment of IP laws rather than their subversion.

There are various reasons why something may not be subject to IP laws. First, the IP laws that pertain to a particular work can expire. So, for example, copyright subsists in a literary work for between fifty and seventy years following the death of the author.[84] Differing time periods exist for different forms of IP and in different jurisdictions, with perpetual protection being possible.[85]

Second, certain types of information are not considered protectable subject matter, therefore, a particular IP regime may not apply. Under European patent law, for example, 'programs for computers' are not considered inventions (see further Chapter 5, at section 5.1).[86] While under US copyright law, works of the US government are not protectable.[87] Copyright also protects forms of expression, rather than the underlying ideas and principles that generate that expression. As such, ideas fall outside international[88] and national[89] copyright regimes and access to such ideas may require specific statutory protection, as provided for in respect of computer software under European Union (EU) law.[90]

Public domain must also be distinguished from exceptions that are carved into IP regimes. With the latter, the right subsists in the information, but the right holder is prevented from exercising that right against a particular use made of that information. In the case of software, for example, European law recognises various exceptions that permit a lawful user to use the software for error correction or back-up purposes.[91] Specific provision is also made for a lawful user to obtain protected information i.e. 'necessary to achieve the interoperability of an independently created computer program with other programs',[92] which was designed to stimulate competition in the software market.

Use exceptions may be drafted broadly, such as the US concept of 'fair use',[93] or narrowly list-specific usage scenarios or purposes, as provided for under EU law.[94] Copyright exceptions are a topic of ongoing political debate in many jurisdictions,

[83] Boldrin and Levine, *Against Intellectual Property*, see note 37, at 17.

[84] The Berne Convention provides for fifty years (art 7(1)), while UK law provides for seventy years (CDPA 1988, s 12(2)).

[85] That is confidential information, as long as it remains secret; trademarks, provided the registration is maintained and it does not lose its distinct characteristics; and database right, where a substantial change or investment is made to the contents.

[86] European Patent Convention, art 52(2)(c).

[87] 17 USC § 105.

[88] For example, TRIPS Agreement, art 9(2); Copyright Treaty, art 2.

[89] For example, 17 USC §102(b): 'In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work.'

[90] Software Directive, at art 5(3).

[91] Software Directive, at art 5(1) and (2). See similarly 17 USC § 117.

[92] Software Directive, at art 6(1).

[93] 17 USC § 107.

[94] Information Society Directive, see note 23, at art 5.

revolving around what constitutes the right balance between the competing public interests and rights of control and access. These arguments are, in part, about what constitutes the proper scope of the public domain, although taking place firmly within the paradigm of copyright control.

Finally, public domain works should also be distinguished from so-called orphan works, where it is impossible to identify the copyright owner, but which are still subject to copyright and therefore constrained from being freely used.[95] Within the software industry there is another variation of the orphan work, so-called abandonware. Here the software remains protected by copyright but the owner is no longer interested in the code, providing no support or other related input, and not interested in policing or enforcing against violations of his copyright.[96] Reasons for abandonment vary, but can obviously include the owner going out of business.

The initial question, why not place source code in the public domain, generates two further questions. First, does the applicable IP regime enable a rights holder to place protected subject matter in the public domain; in other words can they shed the source code of its protective legal coating? Second, if source code can be placed in the public domain, what implications does this change of status have in terms of the ceding a person's ability to control subsequent users of the code?

In respect of the first issue, the problem is noted in the Creative Commons CC0 Public Domain Dedication: 'many legal systems effectively prohibit any attempt by these owners to surrender rights automatically conferred by law, particularly moral rights, even when the author wishing to do so is well informed and resolute about doing so and contributing their work to the public domain'. US copyright law recognises the concept of abandonment, which can be argued as a defence to a claim of infringement. It requires a defendant to show that the copyright owner intends to surrender his rights in the work and has overtly acted in a manner evidencing such intention.[97] Whilst this may prove a substantial hurdle in the case of orphan works, such intent could be easily manifest in an Open Source context through appropriate notices dedicating the work to the public. No similar doctrine of abandonment clearly exists under English law of copyright.[98] While in European civil law jurisdictions, the doctrine appears to be generally unacceptable.[99]

[95] See Directive 2012/28/EU 'on certain permitted uses of orphan works' (OJ L299/5, 27.10.2012).

[96] See Dennis Khong, 'Orphan Works, Abandonware and the Missing Mark for Copyrighted Goods' (2006) 15 *International Journal of Law and Information Technology* 54–89, at 54.

[97] *National Comics Publications, Inc. v Fawcett Publications, Inc.* 191 F.2d 594, 90 USPQ 274. See M W Turetsky, 'Applying Copyright Abandonment in the Digital Age' (2010) 19 *Duke Law & Technology Review* 22.

[98] Philip Johnson, '"Dedicating" Copyright to the Public Domain' (2008) 71 *Modern Law Review* 587–610. Also *Copinger and Skone James on Copyright* (London: Sweet & Maxwell, 2012), at 6–88.

[99] Emily Hudson and Robert Burrell, 'Abandonment, Copyright and Orphaned Works: What Does It Mean to Take the Proprietary Nature of Intellectual Property Rights Seriously?' (2011) 35 *Melbourne University Law Review* 971–1004.

The difficulties in abandoning copyright is in stark contrast to the treatment of other IP rights, especially the registered rights, patents, and trade mark,[100] as well as moral rights discussed earlier. As well as a statutory recognition of surrender, patent rights are also vulnerable to community practices, such as defensive publication, which can undermine the secrecy required when applying for the patent (see further Chapter 10).

An alternative would be for the copyright owner to grant a licence to the world, without any restriction on use. Such a licence remains revocable at the copyright holders will, however, except where constrained by estoppel.[101] The ability to revoke would enable a community to respond in the event that their source code was being used in an unacceptable manner, although the related complexity and legal uncertainty would represent a significant threshold to the taking of such action.

With regard to the second issue, placing source code in the public domain would enable a user to incorporate the code within another work, thereby essentially re-privatising the code, to the extent that it could not be used except on the terms granted by the new copyright owner. As such, public domain equates to a loss of control, undermining the objectives of the Open Source movement.

1.3.4 Open Source policies

A sometimes intensely political area of computing, it is inevitable that Open Source has come to the attention of politicians and policy-makers. Historically, politicians in the US and Europe have been highly supportive of IP laws and the need to strengthen existing rules to reflect the shift to service-based, information-led economies in a rapidly evolving digital environment.

At the same time, however, governments have become increasingly attracted by Open Source for various reasons. First, as users of ICTs, the public sector has often experienced significant disappointments with the deployment of ICTs designed to achieve more efficient and cheaper government. Some have seized upon Open Source as a means of addressing these past failures, based on assertions about its technical superiority and its cost advantages. Second, there is a general desire to stimulate innovation within national economies and Open Source is viewed as contributing to that objective. Third, the dominance of certain market players, particularly from the US, has raised concerns about the competitive position of domestic software industries, which may be bolstered by the adoption of Open

[100] Patents Act 1977, s 29 and Trade Marks Act 1994, s 45.

[101] Johnson, '"Dedicating" Copyright to the Public Domain', see note 98, at 607. Additionally, more esoteric mechanisms may potentially be used, such as the copyright holder executing a deed poll, or entering into a contract with another party under which a licence is granted to 'everyone' as a class of third party beneficiary.

Source.[102] Finally, the trend towards more open government, in terms of transparency, such as freedom of information legislation, has chimed with the concept of Open Source and its 'transparency of process'.[103]

International organisations have embraced Open Source. The United Nations Educational, Scientific and Cultural Organization (UNESCO) has noted that Open Source can play a significant role in ensuring attainment of the UN's Millennium Development Goals.[104] In terms of national, regional, or local government policies towards Open Source, the Center for Strategic and International Studies (CSIS) carried out surveys of published policies between 2002 and 2010, which it groups into four categories:[105]

- Research and development (R&D) -related initiatives, such as encouraging the formation of Open Source development communities;
- Awareness and advisory initiatives, where Open Source is brought to the attention of communities of users, again usually the public sector[106]
- Granting preferential treatment for Open Source; and
- Mandating the use of Open Source by public administrations.

The latter two policy categories are variants that directly increase the adoption of Open Source within the public sector. Over the period, adoption was the most prevalent policy approach, a finding confirmed in another survey of European initiatives.[107]

In general, preferential treatment has targeted the procurement of Open Source-related ICTs ('inbound preference'), ranging from favourable treatment in procurement processes to direct financial subsidy where Open Source is adopted.[108] In some jurisdictions, Open Source has also been adopted as the preferred approach for the dissemination of public sector developed code ('outbound preference'). In 2007, for example, the European Commission approved the 'European Union Public Licence' for the purpose of distributing its own software under a private law arrangement that corresponded with the requirements of European law.[109]

[102] See Hal Varian and Carl Shapiro, *Linux Adoption in the Public Sector: An Economic Analysis* (mimeo, Berkeley, CA: University of Berkeley, 2003).

[103] OSI Mission Statement: <http://opensource.org/about> accessed 21 July 2022.

[104] <https://en.unesco.org/freeandopensourcesoftware> accessed 21 July 2022.

[105] See the March 2010 version: <http://csis.org/files/publication/100416_Open_Source_Policies.pdf>. In 2010, some 364 Open Source initiatives were identified from public sources.

[106] See, e.g., the European Commission's 'Joinup' initiative: <http://joinup.ec.europa.eu/> accessed 21 July 2022.

[107] Stefano Comino, Fabio Manenti, and Alessandro Rossi, 'On the Role of Public Policies Supporting Free/Open Source Software' in K. St Amant and B. Still (eds), *Handbook of Research on Open Source Software* (IGI Global, 2007) 412–27.

[108] Comino, Manenti, and Rossi, 'On the Role of Public Policies Supporting Free/Open Source Software', see note 107.

[109] See <https://joinup.ec.europa.eu/collection/eupl/introduction-eupl-licence> accessed 21 July 2022.

However, as with R&D initiatives, promoting the use of particular licence terms for 'publicly' developed or funded software may itself generate controversy, particularly when choosing the use of copyleft rather than more permissive Open Source licences.[110]

Various tools may be used by governments to facilitate Open Source, especially public procurement procedures and an 'open' standards policy (see further Chapters 21). The former is a demand-side competition measure, given the purchasing power of the public sector. The latter can improve supply-side competition, by facilitating interoperability between devices, software, and data. While Open Source does not equate with 'free', as in no payment or charge, payment issues do arise in the area of standards and patents (see Chapter 16), where there is an ongoing and very topical debate about whether existing royalty-bearing or mandated royalty-free (RF) fair, reasonable, and non-discriminatory (FRAND) licensing arrangements discriminate against either the proprietary or Open Source community, resulting in a market failure that justifies government intervention.[111]

It has been noted that one element in the adoption of pro-Open Source national policies has been anti-Americanism. The CSIS suggests that trends in Open Source policies may reflect market developments in the proprietary software market. So, for example, the launch of Windows Vista in 2006–07 and the resultant criticism and negative press coincided with a rise in the number of published policies.[112] However, companies like Microsoft have been seen to have undertaken a shift from their anti-Open Source stance of a decade ago to being amongst the biggest contributors to Open Source today. This can at least partly be attributed to the rise of Kubernetes and interoperable Cloud Native Software which underpins cloud computing and the platform economy alongside the vast increase in Open Source adoption facilitated by GitHub and other repositories.

In terms of implementation, R&D and advisory policy initiatives generally arise through decision-making within public administrations, which is likely to reduce the political capital required for their approval. By contrast, adoption initiatives, particularly through mandation, will often require, or take, a more 'legal' route, through legislative or regulatory measures. Indeed, proposals for mandation are usually instigated within national or local legislatures, which increase the possibility of political and legal challenge. The CSIS survey indicates that the failure rate is considerably greatly for adoption initiatives, with mandation measures experiencing more failures than approvals.[113]

[110] See Lawrence Lessig, 'Open Source Baselines: Compare to What?' in R W Hahn (ed), *Government Policy Toward Open Source Software* (Washington, DC: Brookings Institution Press, 2002) 50–68, at 64 *et seq*.

[111] See, e.g., M Välimäki and V Oksanen, 'Patents on Compatibility Standards and Open Source—Do Patent Law Exceptions and Royalty-Free Requirements Make Sense?' (2005) 2(3) *SCRIPTed* 397–406, at 397.

[112] CSIS, see note 105.

[113] CSIS, see note 105.

Pro-Open Source policies have inevitably generated controversy and a response from the software and wider ICT industries, generally pitching the proprietary rights holders against the Open Source community. Concerns have also been raised by academic commentators that such policies can represent 'industrial policy by stealth'.[114] To a degree, the issues are analogous to debates in other sectors, especially the utility industries, about the best means of achieving open and competitive markets: does establishing a 'level playing field' require some form of preferential or discriminatory treatment when overcoming certain entrenched market structures?

1.4 'Open' What?

While the previous sections identified some of the philosophical and political dimensions that underpin debates about Open Source, they do not provide a complete description of what 'open' means in terms of its distinguishing characteristics. At an abstract level, 'open' can be defined positively in terms of the freedoms users are granted to use, modify, and share something; as specified most clearly in the 'four freedoms' of the Free Software Foundation (FSF) (see further Chapter 2).[115] Alternatively, 'open' can utilise more negative connotations, through requirements designed to prevent certain behaviours and attempts to exert control, examples of which can be found in the OSI's OSD.[116] 'Open' often equates to accessibility and transparency. Source code should be made accessible for examination and scrutiny by others to enable the ideas and principles that comprise its design and functionality to be discerned and peer reviewed, without necessarily involving any further 'use' in the form of interaction. Although 'free' and 'open' are seen as denoting difference in an Open Source context, since cost is often an element in determining whether something is accessible, 'free' as in 'free beer' often comprises an aspect of what it means to be 'open'. 'Open' can also imply freedom of choice and conduct, facilitating adoption, take-up, and use, as much as rejection and the utilisation of alternatives. Universality is also a connotation of 'open', which links to issues of standardisation and interoperability, critical issues for the software industry and examined elsewhere in the book (see Chapter 11).

The Open Source movement relies upon licences, copyright and patent law to enable the *use* of source code by others, specifically its *modification* and *redistribution*. While 'use' is obviously a catch-all term, as well as a synonym for

[114] Josh Lerner and Mark Schankerman, *The Co-Mingled Code* (Boston, MA: MIT Press, 2010) at 197.
[115] <http://www.gnu.org/philosophy/free-sw.html> accessed 21 July 2022.
[116] For example, prohibitions on discrimination against persons, groups, or fields of endeavour. See <http://opensource.org/osd-annotated> accessed 21 July 2022.

copying in a digital environment, the focus on acts of modification and redistribution are key to the control expressed in licences. A licensor is usually concerned with how a licensee uses the code in two circumstances: where the licensee redistributes the code, or where it is modified and then redistributed. The licensor will want to govern the conduct of users downstream from the licensee as much as licensee himself, liberating or restraining depending on your perspective!

Each of these forms of conduct, use modification and redistribution, can raise concerns for the original creators. As in many areas of law, uncertainties and disagreements can exist about the precise meaning of terms used in statutory copyright regimes, both at a national level and from their interaction in a multi-jurisdictional environment. Language is imbued with cultural and historical meanings that find expression through law and legal interpretation. Private law mechanisms can therefore be a tool to address such uncertainties, either building on the existing framework, filling the gaps, or creating an alternative language. The free software movement, particularly through the GPL, has embraced the latter approach, using terms and defining concepts that are deliberately disassociated from those commonly found within copyright law:

> Over the years, we learned that some jurisdictions used this same word in their own copyright laws, but gave it different meanings. We invented these new terms to make our intent as clear as possible no matter where the license is interpreted. They are not used in any copyright law in the world, and we provide their definitions directly in the license.[117]

This attempt to liberate Open Source from national and copyright law prejudices, whilst deliberately remaining firmly within the jurisdiction of these public law regimes, obviously generates its own challenges and uncertainties for developers and users, as evidenced by the ongoing, sometimes fiercely argued, debates within the Open Source community.

The following sections briefly examine the concepts of modification and redistribution within copyright law and some of the implications and debates within the Open Source community surrounding each concept. Although substantially harmonised, national copyright laws retain enough particularities and peculiarities to render coverage of all jurisdictions impossible. As such, the analysis focuses on US, UK, and EU copyright law.

[117] 'Why did you invent the new terms "propagate" and "convey" in GPLv3?' in 'Frequently asked questions about the GNU licenses' at <http://www.gnu.org/licenses/gpl-faq.html#WhyPropagateAndConvey> accessed 21 July 2022.

1.4.1 Modifications

Modifying source code is an exclusive right granted a right holder.[118] What constitutes modification however is much less obvious, varying in terminology and scope between jurisdictions. The act of modifying source code will also generally involve an act of reproduction, which begs the question whether these purportedly distinct rights are effectively inseparable. However, it is widely assumed or accepted that the distinction has important implications, not least by the Open Source community.[119] It is therefore necessary to examine the concept in an Open Source context.

As most Open Source licences originate in the US, we start with the term 'derivative work', which is widely used and is statutorily defined as:

> a work based upon one or more preexisting works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which a work may be recast, transformed, or adapted. A work consisting of editorial revisions, annotations, elaborations, or other modifications which, as a whole, represent an original work of authorship, is a 'derivative work'.[120]

This is an elaborated version of the definition used in the Berne Convention.[121] Under English law, the restricted conduct is the making of an 'adaptation', with the term being given a specific meaning in respect of a computer program, as 'an arrangement or altered version of the program or a translation of it',[122] which originates in EU law.[123] However, adaptation is more narrowly conceived than the US concept, which generates its own uncertainty when transplanting US-originating licences into an English law context.

A derivative work is granted a new and distinct copyright under US law, although to be derivative, the new work must substantially copy the original and must involve more than a minimal contribution to the original.[124] A derivative work should also be distinguished from an original work that derives only its ideas from another work. To create a derivative work requires consent from the original owner, which is granted under an Open Source licence, subject to conditions such

[118] For example, 17 USC § 106(2) and Software Directive at art 4(1)(b). Note that this right is not harmonised in the EU for other types of work (see Information Society Directive, see note 23).

[119] See generally Lothar Determann, 'Dangerous Liaisons—Software Combinations as Derivative Works? Distribution, Installation, and Execution of Linked Programs under Copyright Law, and the GPL' (2006) 21(4) *Berkeley Technology Law Journal* 1421–98, at 1421.

[120] 17 USC § 101.

[121] Art 2(3): 'Translations, adaptations, arrangements of music and other alterations of a literary or artistic work shall be protected as original works without prejudice to the copyright in the original work.'

[122] CDPA 1988, s 21(3)(ab). At (4) translation 'includes a version of the program in which it is converted into or out of a computer language or code or into a different computer language or code'.

[123] Software Directive, at art 4(1)(b).

[124] Melville Nimmer and David Nimmer, *Nimmer on Copyright* (US: Matthew Bender) at § 3.01 and § 3.03[A].

as paternity notices or contribution back. However, some licence schemes permit copyright owners to refuse by default to allow derivative works to be created.[125] If the licence conditions are breached, the consent is withdrawn and the owner of the derivative work can no longer distribute the whole work but could (theoretically) continue to distribute his contribution. As such, a derivative can be seen as residing somewhere between a joint work, where the work is viewed as an undivided whole (see further at section 1.5 of this chapter), and a collective work, where ownership in the parts are distinct from ownership in the whole.

In a software development context, the focus is on the nature of the interaction between the component source code written by the various contributors. Is the contributed code 'based upon' an existing work? If it is, then is the contributed code sufficiently substantial and original to create a derivative work? If it is not, then is the contributed code sufficiently original to constitute an original work in its own right, which can then be assembled with other such works to form a compilation or collective work?

One central and highly charged debate within the Open Source community, and beyond, concerns the concept of 'linking' and the legal consequences when Open Source code interacts with proprietary code through usage. Linking is a normal feature of programing and usually refers to the interaction between a program and so-called library code, which provides reusable functions for multiple and independent programs.[126] Broadly speaking, the nature of the interaction between two linked components may either be static or dynamic, according to the decision of the program designer, the former being generally viewed as an interaction that creates a derivative work while the latter is not.

The term 'linking' is often used in Open Source literature as shorthand for the multitude of different ways in which distinct pieces of code can interact, interoperate, or 'couple' with other code; other methods include remote procedure call (RPC), system calls, and plug-ins.[127] Such interaction matters because where two or more pieces of code are licensed under different terms (whether proprietary or open source) and the resultant work would be considered 'derivative' or similar under copyright law, then uncertainty is generated both about the licence applicable to the resultant work and whether the modification constitutes an infringement of a licence applicable to any part of the contributing code. Within the Open Source community, not all modifications are possible, because of licence incompatibilities, which can prevent two Open Source pieces of code being combined to create a third (see further Chapters 3 and 4). Where a contributing licence is 'copyleft' in nature, such as GPLv2, then the resultant work may have to be made

[125] For example, Creative Commons 'Attribution-NoDerivs 3.0 unported'.

[126] See <http://en.wikipedia.org/wiki/Library_(computing)> accessed 21 July 2022.

[127] See the Free Software Foundation Europe, 'Working Paper on the legal implication of certain forms of Software Interactions (a.k.a linking)', available online at <http://www.ifosslr.org/public/LinkingDocument.odt> accessed 21 July 2022.

subject to that same licence, or the terms of the contributing licence will have been breached and will terminate. As such, one result of copyleft licensing is that the operation of copyright law, whether through its application or uncertainties about its application, has led to the use of software development techniques designed specifically to minimise the risk of any interaction triggering a legal consequence.

Indeed, hardware controls have also been developed and deployed specifically to constrain the effective operation of copyleft licences. TiVo, the producer of digital video recorders, utilised Linux and GNU software within their device, but designed the system to use digital signatures such that modified versions of the source code would not run on the device as the digital signatures would not match. The validity of this approach generated significant controversy within the Open Source community, with some, particularly the FSF, viewing such 'TiVoisation' as unacceptable,[128] while others, such as Linus Torvalds, viewed it as a legitimate business practice.[129]

As with much in law, the answer to these uncertainties will depend on a range of factors, specifically the technical nature of the interaction taking place, the person causing the modification to occur, the jurisdiction in which such modification takes place, and the applicable licence. First, all computer code is designed to interact at some level with something else, whether other code, hardware, or otherwise. As such, 'mere' interaction or interoperation between codes is not sufficient to render the outcome either a work or a derivative work. Works may interact but remain distinct and separable, each its own copyrighted work. The works may be used together and be redistributed as a package, but remain distinct within a collective or composite work,[130] also referred to as 'mere aggregation'.[131] Second, the end-user receiving the composite work may create a derivative work for his own purposes, without further redistribution. As such, the end-user's conduct may differ from the intermediary distributor because the conditions of the licence are only triggered by an act of modification *and* redistribution. Third, the copyright law of the jurisdiction in which the interaction takes place may interpret what constitutes a derivative work differently from its neighbouring jurisdictions, whether more narrowly or broadly. Finally, while the wording of any applicable licence may not survive judicial review under either a copyright or contractual analysis, a licensee is generally advised to give due consideration to such wording, which may differ in important respects from the governing legal framework, particularly when adopting a broad interpretation of what constitutes modification. The GPLv2, for example, governs not only derived works, but works that 'in whole or in

[128] See <http://www.gnu.org/licenses/gpl-faq.html#Tivoization> and GNU GPL v3, at 6, paras 4–5.

[129] See <https://groups.google.com/forum/?fromgroups#!topic/fa.linux.kernel/L5NRD_ONkIk>.

[130] For example, Berne, see note 49, at art 2(5).

[131] See the GPL FAQs, at <http://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html#MereAggregation> accessed 21 July 2022. See also the GPL v3, at 5.

part contains'[132] the licensed code, which would appear to include collective works where the distinct copyrighted works may interact only in a minimal way.

1.4.2 Distribution

Under the WIPO Copyright Treaty (1996), distribution is recognised an exclusive right of an author: 'Authors of literary and artistic works shall enjoy the exclusive right of authorizing the making available to the public of the original and copies of their works through sale or other transfer of ownership.'[133] Under EU law, the distribution of computer programs to the public is one of the exclusive rights granted to the right holder.[134]

While an act of distribution extends both to the original work and copies, it is generally only engaged where copying is involved. Under traditional copyright principles, where a copy of a work is redistributed, without a further copy being made, then the copyright owner is constrained from prohibiting such conduct under the doctrine of 'first sale'[135] or 'exhaustion'[136] of the distribution right. The historic rationale for this doctrine is that the copyright owner should be remunerated for the copy but not for any further economic value derived from its further sale down a chain of consumers.[137] While the doctrine refers to 'sale', it is in fact applicable to other situations where the copy is passed on to others, whether for remuneration or otherwise.[138] In the European Union, the doctrine is also used a tool to promote the single market and prevent market partitioning,[139] which fundamentally distinguishes its application from that in the US.

In the US, the exhaustion doctrine has been held not to apply to pure 'digital works' both by the courts and the relevant authorities.[140] In addition, the courts have specifically held in relation to software licences that where the copyright holder clearly indicates that the user is a licensee, restricts the user's right to transfer the licence, and restricts the use made of the software, the first sale doctrine is not

[132] GNU GPLv2 (1991), at 2(b).

[133] Art 6(1).

[134] For example, Software Directive, at art 4(1)(c).

[135] 17 USC § 109(a).

[136] Copyright Treaty, art 6(2).

[137] Under EU law, such remuneration should also be that which is 'appropriate', rather than the 'highest possible remuneration'; see *Football Association Premier League Ltd and others v QC Leisure and others, Murphy v Media Protection Services Ltd* [2012] 1 CMLR 29, at paras 108–109.

[138] For example, in the UK, the CDPA 1988, s 18(3), refers to a loan.

[139] For example, Case C-200/96 *Metronome Musik* [1998] ECR I-1953, para 14. This principle is limited to distribution within the EEA and does not apply internationally (see *Laserdisken ApS v Kulturministeriet*, Case C-479/04, [2007] 1 CMLR 6, at para 24).

[140] See *Capitol Records LLC v ReDigi Inc.*, No. 12 Civ. 95 (RJS), 30 March 2013; also US Copyright Office, DMCA Section 104 Report (August 2001), at 97 *et seq*.

applicable.[141] The doctrine could apply to a software transaction, however, were the circumstances such that a licensor/licensee relationship was not successfully established, but title in the software was held to have transferred instead.[142] In the absence of evidence of an agreement or conduct indicating acceptance by the user, such so-called shrink-wrap, 'label', or unilateral licences may not be considered enforceable, which equates to the question of whether Open Source licences are considered contracts or not (see further at Chapter 3).

Under EU law, the exhaustion doctrine is expressly extended to computer programs,[143] although until recently its application was widely seen as being restricted to the distribution of tangible copies. This limitation was recently rejected by the ECJ in *UsedSoft GmbH v Oracle International Corp* (2012).[144] Here, Oracle made client-server software available for downloading from a website free of charge, but subject to a usage licence. Oracle offered group licences that permitted up to twenty-five users, while if a licensee had more users, it would have to obtain another twenty-five-user licence. UsedSoft obtained these group user licences from Oracle's customers and offered any unused user permissions for sale to others, which Oracle considered to be an infringing act. The Court held that where a copy of the program was transferred to a user, whether through a tangible medium such as a DVD or made available for downloading from a website, together with a licence granting a right to use the program for an unlimited period, then that constituted a 'first sale' for the purpose of the exhaustion doctrine (para 49).[145] Quoting approvingly the Advocate General's opinion, to distinguish a contract as being either a licence, to which the exhaustion doctrine does not apply, or a sale, to which it does, would be to undermine the purpose of the provision itself (para 49). Previously, it had been widely believed that making software available for download was an act of 'communication to the public', which is a different exclusive right granted the right holder and one to which the doctrine of exhaustion does not apply.[146] However, the Court held that the transfer of ownership or 'sale' that resulted from the downloading of a copy and the granting of a licence to use rendered the conduct within the scope of the distribution right (para 52).

One implication of this decision is likely to be to encourage licensors to alter their distribution model, shifting away from a 'sale' business model towards a 'rental' subscription model, which also reflects an industry trend towards SaaS

[141] *Vernor v Autodesk, Inc.*, 621 F.3d 1102, C.A.9 (Wash).), 2010. See also *Apple Inc. v Psystar Corp.*, 658 F.3d 1150, C.A.9 (Cal.), 2011 and *MDY Industries v Blizzard Entertainment*, 629 F. 3d 928 C.A.9 (Ariz.), 2010.

[142] *UMG Recordings, Inc. v Augusto* 628 F.3d 1175 (9th Cir. 2011), which involved the distribution on digital content on physical CDs. See also *SoftMan Products Co., LLC v Adobe Systems, Inc.* (2001) 171 F.Supp.2d 1075.

[143] Software Directive, art 4(2).

[144] 3 CMLR 44.

[145] Oracle's licence stated that it was 'non-transferable', but this was effectively ignored by the Court.

[146] See Copyright Treaty, art 8 and Information Society Directive, see note 23, art 3.

and cloud computing (see further Chapter 9). For the Open Source community, however, cloud itself can be seen as an alternative mechanism for restricting the freedom of users to modify the software they use and depend on, controlling rather than liberating.

Under Open Source licences, redistribution is not simply about copying the actual code but also about the conditions under which the recipient receives the code, either requiring the original rights to be matched throughout the distribution chain ('copyleft') or enabling the substitution of different rights for subsequent users, which may be more restrictive. The term 'viral' has been used to describe the manner in which certain Open Source licences operate, also referred to as 'copyleft', by imposing obligations down the software distribution chain.[147] While all commercial agreements attempt, to some degree, to ensure that obligations are appropriately reflected either upstream or downstream, the description of copyleft licences as 'reciprocal' (sometimes 'viral', although the Open Source community tends to avoid the term 'viral' owning to its derogatory connotations) arises from the self-executing nature of the licence, relying on copyright law rather than contract. Other examples of 'viral' laws are some export control regimes, which apply to specific applications such as encryption modules within devices, but can operate such as to make the whole product subject to, or 'infected' by, the control regime.

While the right to 'distribute' is commonly used within copyright regimes, concerns about jurisdictional differences about its scope has seen the FSF deploy the term 'convey' in the GPL, defined as follows: 'a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.'[148] The second part of this definition is important in the context of cloud computing and is examined elsewhere (see Chapter 9).

Under international, regional, and national copyright rules, the concept of distribution is qualified by the phrase 'to the public', which suggests the possibility of non-public offerings of copies of a work. Understanding where the boundary exists between the two can obviously be important in the context of Open Source development. The actual numbers of persons in receipt of a work, while a factor for consideration, is likely to be less determinative than the manner in which the copies of the work were made available; 'what counts is the general *opportunity* given to the public'.[149] The ECJ, in *Sociedad General de Autores y Editores de Espana (SGAE) v Rafael Hotels SL*,[150] has noted in respect of the right to communicate a work to the

[147] For example, Andrés Guadamuz Gonzalez, 'Viral Contracts or Unenforceable Documents? Contractual Validity of Copyleft Licences' (2004) 26(8) *European Intellectual Property Review* 331–9.

[148] GNU GPLv3 (2007), at 0.

[149] Sam Ricketson and Jane Ginsburg, *International Copyright and Neighbouring Rights* (Oxford: Oxford University Press, 2006), at 11.91.

[150] Case C-306/05, 7 December 2006; [2007] ECDR 2. See also Case C-607/11, *ITV Broadcasting Ltd & ors v TVCatchup Ltd*, 7 March 2013.

public, that 'it is sufficient that the work is made available to the public in such a way that the persons forming that public may access it' (para 42). This has potential implications for the governance of Open Source projects, since the more restrictive the conditions of participation are made, the stronger the argument that the distribution of code among participants does not constitute an exercise of the exclusive right in respect of the code, which could trigger certain consequences under the applicable licence. Conversely, restrictive membership runs counter to the open collaborative model that underpins Open Source.

The 'to the public' qualification is absent from the concept of 'conveying' used in the GPL v3, which raises the question whether certain forms of distribution, whilst not a breach of copyright law per se, may result in a breach of a licence? In a development context, for example, code may be distributed to participants within a community or forum, which extends beyond a single organisation, or demo code may be given to certain selected customers for the purpose of testing under a non-disclosure agreement (NDA). The restricted nature of the distribution would not appear to be 'to the public', yet it could give rise to a technical breach of the GPL because an 'other' party has received a copy. The GPL v3 does provide that conveyance may be to 'others' but only where it is carried out 'exclusively on your behalf, under your direction and control',[151] which recognises the non-public nature of the distribution. However, it also states that such conveyance is only permitted for the 'purpose of having them make modifications exclusively for you, or provide you with facilities for running those works', which would suggest an employer/employee, principal/agent, or customer/supplier-like relationship between the parties, but would not necessarily cover either the community distribution or the 'testing' scenario (see also Chapter 9, for its application in a cloud context).

As distribution of the source code and any accompanying text is the central behavioural obligation placed on users, disputes have inevitably arisen about whether this obligation has been properly met, particularly in the context of retail products.[152] The code may be distributed on some associated media distributed with the product, or may be offered to the end-user, often in the form of a web-based download. In either case, but often in the latter, uncertainties can arise as to whether the availability or offer of the code is made *sufficiently* transparent to the end-user to meet the licence requirements. Such issues are analogous to other requirements in law concerning notice, such as the contractual incorporation of terms, and availability, such as the decompilation obligation.[153] Retail products often generate particular issues where marketing, design. and brand concerns are to the fore.

[151] GNU GPLv3 (2007), at 2.

[152] For example, *Welte v D-Link Deutschland GmbH* (2006) LG Frankfurt a.M., 2006-09-06, Case No. 2-6 O 224/06.

[153] Software Directive, art 6(1)(b), which restricts the decompilation right where the necessary information has been made 'readily available'. Similar wording is used in the US, at 17 USC § 1201(f)(1).

As with modifications, by focusing on the concept of distribution as the trigger for certain legal consequences, whether desirable or otherwise, attention inevitably converges on the meaning of the term, with, as in much of law, plenty of scope for argument and debate. The traditional venue for interpretation is the courts, although there is scant directly applicable guidance available to date. The licensor is left with the option of trying to draft appropriate and sufficiently precise provisions to address any uncertainties; a complex task in such a rapidly developing environment.

1.5 Open Source as Development Methodology

Software development or engineering has evolved considerably since the early days of computing, as processing capacity and programing languages have enabled ever more sophisticated systems to be developed. Greater sophistication of the end product, the source code, has also required more formalised development processes, in order to address the needs of users adequately, reflect those needs in feature design, and test and verify the resultant product. Concomitant to these developments, we have seen the industry try to professionalise itself, establishing qualifications and standards which programers can obtain and meet. Various standards and development methodologies have been promulgated to capture the various stages of software lifecycle and, thereby, improve the quality of source code,[154] such as the 'Waterfall' model, Spiral, and Agile. The latter is seen as the most widely adopted methodology with the Open Source community (see further Chapter 7).

Open Source communities can also be seen as a development methodology in their own right, as noted by the OSI:

> Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in.[155]

1.6 Open Source as Commerce

One recurrent theme has been to acknowledge the philosophical and political ideas that underpin the Open Source movement whilst at the same time treating

[154] For example, ISO/IEC 12207: 2008 *Systems and software engineering—Software life cycle processes*.

[155] OSI Mission statement: <http://opensource.org/about> accessed 21 July 2022.

Open Source apolitically, i.e. as a reality that organisations need to be aware of, to act appropriately towards, and consider as an alternative means of doing business.

Awareness is obvious, yet organisations, as much as individuals, exhibit a tendency to turn a blind eye to those things which are not well understood or seem difficult. In the author's experience, while the existence, development and deployment of Open Source is widely known and understood, there is a considerable lack of knowledge and uncertainty about the legal implications of using Open Source and its good governance.

Acting appropriately, once aware, requires consideration of how Open Source may impact the organisation both internally and externally, with regard to suppliers, partners, and customers. Internal considerations include regulating employee contributions to projects and use of Open Source, while external issues include auditing for Open Source in due diligence procedures (see further Chapter 8). A key objective of most organisations is to manage their risk and to protect their assets collectively known as curation.

There is a variety of ways in which Open Source can be used to generate economic value, through so-called hybrid strategies.[156] The range of possible business models reflects the complexity of the ICT ecosystem (see further consideration in Chapter 16).

1.7 Enforcing Open Source

As discussed throughout this book, the Open Source community relies on copyright law and contract law, and to a greater or lesser extent patent and trade mark law, to govern the conduct of users of their code. As such, user non-compliance with either the terms of a licence or the underlying statutory obligations, gives rise to the possibility of enforcement action being taken against code users, as licensees or otherwise. Indeed, enforcement, or the realistic threat and risk of it, must be seen as an essential component of any effective legal regime, whether based in private or public law, although fear of enforcement may not be the primary reason why laws are respected.[157]

IP laws provide a range of civil and criminal remedies against infringers, explored more fully across the IP Chapters of this book, with different remedies depending on the characterisation of the breach as either being copyright or

[156] See R van Wendel de Joode, H de Bruijn, and M van Eeten, 'Living Apart Together: Hybrid Business Strategies on the Edge of the Commons' in *Protecting the Virtual Commons* (The Hague: Asser Press, 2003) 93–107.

[157] See further Chris Reed, *Making Laws for Cyberspace* (Oxford: Oxford University Press, 2012).

contractual in nature,[158] which generates another layer of uncertainty for the enforcement of Open Source licences.[159]

Enforcement actions may also arise under other complementary legal regimes, such as consumer protection laws designed to prevent the defrauding of end-users. One example is so-called subscription traps, where sites offer Open Source software for prohibitive and hidden fees, which have been the subject of criminal proceedings in Germany.[160]

As well as actions by Open Source rights holders against code users, disputes also arise between rights holders and other rights holders, whether Open Source or proprietary. Given the risks and costs associated with enforcement actions, potential defendants will commonly adopt a range of strategic IP management measures intended to mitigate such risks, from technical 'design-a-rounds' to the defensive acquisition of IP rights (see further Chapter 6, at section 6.9).

1.8 Open Futures

This chapter has identified numerous features of Open Source-related conduct which generate legal uncertainties, uncertainties that can operate to the detriment of both proponents and users of Open Source software. These include the language used in most Open Source licences taken from US copyright law, which can differ in important respects when transplanted into other jurisdictions. Some Open Source licences contain terms which have been drafted to address specific concerns and needs of certain communities, but which may be unfamiliar to many users of the code released under that licence, and which may also be untested before the courts. The collaborative working structures in Open Source communities are also unfamiliar to many and are therefore often ignored or managed poorly, as well as generating complexities for the application of the law.

Certain developments may also weaken the influence of current Open Source licence arrangements over time, particularly those pursuing a copyleft philosophy. In terms of markets, the shift to cloud computing means that software distribution is becoming less important in a SaaS environment, except in the context of operation of the physical access device itself. Evolving copyright law, driven as much through judicial decision-making as legislative reform, may increasingly constrain the application of copyright law to software, reducing its efficacy as a control

[158] In the US, see *MDY Industries v Blizzard Entertainment*, 629 F.3d 928 CA9 (Ariz.), 2010, where the court held that a contractual breach did not have the necessary 'nexus between the condition and the licensor's exclusive rights of copyright'.

[159] See Robert Gomulkiewicz, 'Enforcement of Open Source Licenses: The MDY Trio's Inconvenient Complications' (2011) 14 *Yale Journal of Law and Technology* 107–37.

[160] LG Hamburg, Judgment of 21 March 2012, Az 608 KL 8/11, available at <http://openjur.de/u/432081.html> accessed 21 July 2022.

regime. While from a technical perspective, the use of increasingly sophisticated techniques designed to limit the interaction of software components subject to divergent licensing arrangements may reduce the viral impact of copyleft licences.

As noted at the start, this chapter has tried to avoid making normative statements about Open Source. Those that promote defend and use Open Source may pursue particular philosophical, ethical, or political aims, which are noted and respected. Instead, the focus has been on how public law regimes, particularly copyright law, interacts with Open Source software to facilitate and constrain the aims of Open Source proponents and the use of private law arrangements to achieve specified outcomes, outcomes that can be designed to subvert the public law settlement. All the evidence shows Open Source development and usage increasing substantially. This is likely to result in greater judicial consideration of how the language and law of Open Source operates, hopefully reducing some areas of uncertainty. What is more unpredictable, however, is whether governments and legislators will address the rise of Open Source redesigning areas of public law to reflect the critical role and unique features of software in society and the economy.

2
Evolving Perspective on Community and Governance

Ross Gardler and Stephen R Walli

2.1 Collaboration and Communities

'We've known how community works, since you had a campfire and I wanted to sit beside it.'[1] This is a simple truism about humans, the societies we build, and their success. We have collaborated on software since we started writing software, and this collaboration goes all the way back to the early work on programable computers by von Neumann's team at Princeton University. Writing good software is hard work. All the investments that have been made in computer programing and software engineering in the past seventy years have essentially been about

[1] Stephen Walli, 'What does an adult look like in your community', speaking at the Community Leadership Summit, July 2014. www.communityleadershipsummit.com

Ross Gardler and Stephen R Walli, *Evolving Perspective on Community and Governance* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Ross Gardler and Stephen R Walli 2022.
DOI: 10.1093/oso/9780198862345.003.0002

writing more and better software in fewer lines of 'code'. Developers collaborating in communities, the idea behind Open Source, may well be the best software reuse strategy society has invented.

To come to that idea, we need to consider a little bit of history around software sharing, the nature of copyright and licences, and how these communities form, organise, and govern themselves. Then we can understand better the engineering economics and business implications.

2.2 Intellectual Assets to Intellectual Property

The creation of complex software requires intellectually challenging effort. There was a time through the 1950s, 1960s, and 1970s, however, when software was often created by computer manufacturers and bundled with computers. The cost of the computer itself far outstripped the human cost of developing software for it.

Conferences, now the heart of this sector, began to be organised by computer manufacturers in this period, were opportunities to share software practices, ideas, and the software itself. The IBM conference, started in the 1950s, was in fact called SHARE. DECUS was the user society that sprang up around Digital Equipment Corporation computers. As AT&T Bell Labs began to share tapes of the early UNIX operating system, USENIX emerged as the conference where people began to share software tools and practices based on it.

In 1980, everything changed. Copyright was applied by the US Congress to computer software. Copyright is the legal mechanism historically used to protect the author of code and is discussed in detail in Chapter 3.

It costs money to bring an author's creation to the masses, and copyright law gave distributors the legal framework to protect their investment. Once applied to software, however, what had been intellectual assets that could be freely shared in a community of like-minded users transformed to IP protected assets in a protected distribution system.

In the forty years since copyright was first applied to software, the cost of the computing hardware has plummeted, while the cost and complexity of computer software has increased exponentially. Computer hardware, programing languages, and operating system interfaces have standardised through this period (as both *de facto* technologies and through rigorous *de jure* processes). The creation of the Internet and World Wide Web through standards of hardware, software interfaces, and protocols removed friction from the digital distribution pipeline. Software was no longer printed to media and shipped. Instead, it was distributed and updated digitally, over the airways, and increasingly for free. These trends together have enabled a rich and vibrant computer software industry, separated from the computing hardware, to grow and thrive.

2.3 Intellectual Property and Industrial Scale

Creating a computer program to solve a problem is an intellectual pursuit. If we write a program for ourselves to solve a problem, it requires knowledge in both how to solve the problem algorithmically and the knowledge to express the solution in code so that hardware can execute it. But if we create a computer program that we share with a small group of ten friends, there is additional work required to ensure we can package and deliver the software to them. If that number of friends was to grow to 100, considerably more work is needed to 'maintain' the software package. As with all engineering endeavours, as one scales the number of users of an artifact, one requires more disciplined practices and processes to manage the work efficiently in order to create, package, distribute, and support it at scale.

Software companies became extremely efficient at these engineering-at-scale practices over the years. Through the 1980s and 1990s and into the new Millennium, the software industry grew, selling software protected with its value generated by copyright. Whether the price was considered a licence royalty, a support contract, or evolved into an ongoing subscription is somewhat irrelevant. What matters is the value of the revenue generated. During this period a large number of businesses and governments created enormous volumes of bespoke software in IT departments. There were many problems that can't necessarily be solved 'out of the box' by a piece of bought software. The 'build versus buy' analysis used is familiar to every software company product manager and corporate IT manager.

2.4 Early Experiments under Copyright

This dichotomy (bought versus bespoke software solutions) was the world into which open source software and free software landed and thrived. Although the application of copyright to computer software means a licence is required to grant permission and define the terms of that permission or use of the software's copyright, the idea of sharing and collaborating on code continued. Just as software companies created commercial or proprietary licensing agreements to define the terms of use for their customers, developers who wanted to continue to share and collaborate also needed to create licences to allow third parties to use and collaborate on their work/code due to the application of copyright to code.

Several licensing experiments began in the 1980s and 1990s to enable such collaboration:

- Project Athena at MIT was a collaborative experiment between DEC, HP, and IBM, researchers, and others, leading to the X11 Windowing System and Kerberos. It was the start of the MIT/X11 licensing experiment.

- The early articulation of Free Software ethics began in the early 1980s and led to the ideas of copyleft, a user's rights with respect to software, and the GNU Public License. (We will discuss more about software freedom and ethics later in this chapter.)
- The Berkeley Software Distribution (BSD) begun in the Computer Science Research Group at University of California at Berkeley led to the BSD license variants that began to show up in other collaborative projects like the early Apache web server project.
- The Perl language licensed under the Artistic License was the basis of an enormous community of developers and systems administrators sharing and collaborating on the language and tools written in it.

The industry would not think about these early-stage licences as classes of 'Open Source licences' until the end of the 1990s, but for eighteen years as a consequence of the need for licences under the copyright regime, developers and companies continued to share software through licences, as they experimented. They not only experimented with sharing the code itself but with licences that enabled such collaboration around code, just as they had shared for the previous thirty years before 1980 and the application of copyright to code.

2.5 The Start of an Engineering Economic Model

The last piece of the puzzle to understand before going deeper into the discussion of community and governance is an economic one. Even today, people still question Open Source software at a high level as an idea. They do so because it is freely distributed by developers at zero cost. There is no apparent economic model when viewed in this simplistic way.

An easy way to provide a perspective is to consider a company consuming Open Source licensed projects into the company's software solutions. Every IT manager and software product manager is familiar with the build-versus-buy analysis as discussed as a method of acquiring software as a solution to a problem.

With liberally licensed and collaboratively developed software projects, there is, however, a third choice: borrow and share.

An example to set-up the thinking on this:

- In a small startup in the late 1990s,[2] the engineering team needed control of their compiler environment building kernel level software on Windows NT

[2] An author (Walli) was VP, R&D of Software Systems, building the UNIX front end on Windows NT 1995–1999. This is a direct example of the costs and decisions around the GCC compiler suite made to improve the product. Much of the product was developed out of liberally licensed, collaboratively developed software projects that would not be called 'open source' for a couple more years.

across three architectures (IA-32, DEC Alpha, MIPS). The Microsoft compilers were insufficient. Buying the Intel compiler would not solve for the other two computing architectures. The build-versus-buy decision did not meet the startup company's needs.

- The GCC compiler suite provided an answer. This collaborative project was mature and vibrant and ran across the three computing architectures required.
- For approximately US$100,000 investment in hiring a compiler engineer, the company captured US$10 million in value from the GCC project[3] in the first year across all three computing architectures.
- Now, however, the startup was living on a modified fork from the main GCC project. This meant they were cut off from easy access to bug fixes, new performance improvements, and new functionality.
- The startup spent an additional US$40,000 on compiler contractors who were maintainers in the GCC projects and had their startup changes re-integrated upstream into the GCC project head revision.
- This meant that with the next release from the GCC project, the startup would be able to directly use the release at minimal integration cost because their changes were already present in the main GCC project, as well as gaining access to all the new work in community.
- Integration costs for new GCC releases into their product going forward were then in the order of US$10,000 per major release. The new value built into GCC over those eighteen months by the broad GCC community was on the order of an additional US$5 million. This is three orders of magnitude of value capture over the eighteen months.

Borrow and share enables additional analysis beyond build versus buy.

It is not enough just to borrow. Software is surprisingly dynamic with a steady flow of changes and bug fixes happening, especially in a vibrant community setting. Sharing back any changes required to use the software, adding these to the borrowed software, and giving these to the community or project that created the original software borrowed ensures easier engineering access to the new and fixed functionality on a go-forward basis.

It is easy to see why you would use such existing components if they were available to borrow freely and were of sufficient quality. However, the question remains as to why one might create such a component project in the first place? Why create this and do the additional work to build and manage a community around a freely

[3] Using a COmparative COst MOdel (COCOMO) calculation on the project size of approximately 750,000 lines-of-code (LoC) in 1997. While every software developer can articulate the problems with LoC as a strict measure of software value, it makes for an interesting relative measure over time, and can be compared to direct salary and contracting costs.

available component. For those answers one needs to look to innovation model research from Eric von Hippel starting back in the 1980s.

Von Hippel first studied the sailboarding community. While there were a number of manufacturers, 'Windsurfer' being the dominant brand name, none of these companies was spending heavily on research and development for new sail-boarding technologies. There was still a vibrant community of sailboard 'hackers' across the community who were cutting, drilling, gluing, and otherwise experimenting with boards, booms, masts, and sails. The companies were happy to stand back and to watch and learn, without necessarily making costly mistakes in bringing to market experiments that did not work or did not capture the community imagination. The sailboard aficionados on the cutting edge were not interested in building companies. They were happy to share their innovations and ideas and to be sponsored, to be the coaches, consultants, and trainers, and to live on the cutting edge.

The further von Hippel investigated this, the more he discovered these innovation models applied in other areas, eventually doing research in the software domain, and directly in the Open Source domain. While one can look to this research for support for the innovation model, one has to remember that commercialisation of software only evolved after the application of copyright and that the original model utilised by the software development community before copyright came along was one of collaboration and sharing.

2.6 Open Source as a Shared Production Model

Open Source is a number of things and one of these is a software licensing, production, and distribution model. As a licensing and distribution mechanism it provides software under terms that allow users selfishly to do as they please with code otherwise protected by copyright. However, as we have seen, it is to that selfish user's advantage to contribute any changes they make back to the project that originally created and distributed the code.

As a means of production, the Open Source model minimises the cost of production through efficient collaboration, amortising the cost of creation across all participants. The Open Source development model allows individuals to bring their specialist and valuable knowledge and share it with others who may be equally skilled. In return for their contributions they receive improved software. The economic justification for contributing is easy to see because a small contribution (relative to the entire project) is rewarded by a more complete solution on a freely available basis.

Open Source allows individuals and organisations that potentially compete in the marketplace to collaborate seamlessly on software components and systems that, once created, are easily shared, avoiding duplication of efforts and investment and potentially building better outputs through diverse collaboration. The unrestricted distribution of such software results in a larger ecosystem of users which in turn increases the number of potential collaborators on a go-forward basis and thus facilitates a further reduction in the cost of production.

The effectiveness of this model of production can be seen in many Open Source projects. Consider, for example, the Apache Hadoop project.[4] This is an implementation of Google's Map Reduce algorithm. Hadoop contains significant contributions from many companies that operate in overlapping markets. Participants include large software companies such as Microsoft, Facebook, Twitter, and LinkedIn as well as small and medium-size enterprises (SMEs), universities, and government organisations. There are even some individuals involved. All contribute on equal terms, regardless of their size, to the production and maintenance of the Hadoop software. Most contribute in order to reduce the costs and increase the quality of software that forms a core part of their unique business models; a few contribute for more personal reasons, such as professional development.

For this model of production to work it is necessary for each participant to realise more value than they contribute. Participants must also feel that their future is protected and that their contributions cannot be abused by other collaborators now or in the future. It is the combination of a community-based development model, backed by modern Internet-based collaboration tools, and an Open Source licence that ensures such collaborations are possible. Of course, they must also set up structures which allow for collaboration within the realms of the antitrust and competition regimes.

2.7 Open Source Culture

When reading about Open Source one will often find reference to 'free sofware or open software community'. This implies a single coherent community that rallies around the Open Source banner and all it represents. However, there is no such community, just as there is no 'proprietary community'. There are, instead, a number of distinct communities who rally around specific software projects, licensing models, and development models to address specific needs. These communities do not form a part of a larger coordinated and coherent 'Open Source community', although they may be related in one or more ways with other subcommunities. There are therefore a number of distinct clusters of communities that for a variety of reasons gather in a single place. The following paragraphs examine some of the common reasons for such clustering.

[4] <http://hadoop.apache.org/> accessed 21 July 2022.

Clustering can simplify the management of common factors across projects. For example, IP and project infrastructure facilities might be provided by and held in a central body. There are multiple places where such gatherings occur, for example the .NET Foundation[5] provides an IP shelter while GitHub and GitLab[6] provide technical infrastructure. This kind of colocation does not, under normal circumstances, lead to the creation of a single unified community across projects.

Projects may also gather together in order to share community management expertise. An example of such a community can be seen in the Apache Software Foundation (ASF) The ASF has a governance model in which it is not possible to buy influence. The only currency of value to the ASF is merit in recognition of productive engagement. This provides a neutral space in which people can openly collaborate. In such clusters cross-project collaboration is more likely but it is a by-product of standardisation on governance models and IP management rather than a requirement of the community structure. The ASF famously operates according to the Apache Way, people before code, exemplifying the importance of the human community.

Other community structures can be developed to enforce cross-project collaboration. This is useful when a number of organisations choose to collaborate on a specific set of common shared software projects. In order to enforce a certain level of commitment to these projects, partners may choose a model in which strategic influence is a reward for adhering to the rules of participation. Those rules may or may not involve an element of directly productive contributions to software code. Such an environment is designed to be less neutral than a pure community model but they still cannot be controlled by a single participant. An example of such an organisation is the OpenInfra Foundation[7] in which two-thirds of the Board of Director seats are essentially 'paid for' while the final third are representatives of the active community regardless of their financial contributions. In these kinds of clusters collaboration levels across projects are high since there is a very tight focus and clear strategy for the products being produced from project components.

In busting the myth of 'the Open Source community' we need to understand that the primary driver for collaboration is to benefit from the outputs of the individual community project rather than to rally behind a generic Open Source banner. In these cases Open Source is nothing more than a means of production, however there is one final type of community that is usually referred to as the free software community.

The free software community feels that open source and its focus on methods of production are less important. Members of this community prefer the term 'free software', rather than 'open source software', as they are concerned with the

[5] <https://dotnetfoundation.org/> accessed 21 July 2022.

[6] <http://github.com> accessed 21 July 2022.

[7] <http://www.openstack.org> accessed 21 July 2022.

provision and protection of software that respects the users' freedom to run, copy, distribute, study, change, and improve the software. The free software community coalesces around the Free Software Foundation (FSF) (see Case Study: The Free Software Foundation) which provides a legal home for free software but it also delivers appropriate support for advocates of the ethical considerations that drive free software supporters.

These ethical considerations are important but are often seen as overstated by more pragmatic open source participants. We will discuss this in more detail in the next section. Having established the importance of this ethical position we will continue to use the term 'Open Source' to mean software that is licensed in such a way that it is considered to be both open source software and free software. Where we wish to make a distinction between the legal protection of IP in software and the ethical considerations of software freedom we will use the term 'free software'.

It can be seen that while there is no single 'Open Source community' there are a great many sub-communities. These communities are linked by one or more of the following characteristics:

- A sharing of a legal structure for IP management
- Sharing of project infrastructure (website, version control, mailing lists, etc.)
- Adoption of an agreed collaborative software development model
- Requirement for a neutral space for collaboration
- A sharing of common needs that can be solved with software outputs
- Enforced collaboration on shared software components
- An ethical belief that all software should be free (as in free speech)
- Collaborative raising of support whether in funding or resource

2.8 Licences to Facilitate Collaboration

A common factor across all of these communities is the adoption of an Open Source licence for their collaborative outputs. Experimentation with copyright law has created the concept of copyleft and a range of licences that protect the IP created by each participant whilst allowing for unrestricted distribution of the code. Today there are many different Open Source licences to choose from and this is discussed in depth in Chapter 3.

Which licence is chosen by a given project will have significant impact on the kind of community, and thus the kind of production model, that a project adopts and plays a surprisingly significant role in community development. As we have seen, earlier licences began around individual experiments in collaboration, but as more people and companies became involved, the licences could be seen to fit into broad categories and the benefits of standardisation became clear.

One group of licences, commonly known as reciprocal or 'copyleft', legally enforce the publication of changes to code, that is, derivative works which incorporate the original code. These licences may impact some business models, such as the creation of closed, proprietary derivatives from the open code but at the same time ensure that no third party can abuse the open development model by simply consuming the projects outputs without publishing their changes (see further Chapter 3). In developing copyleft as a concept Richard Stallman, its creator, played a pun on copyright and protected users from themselves and any desire they might have not to give back.

At the opposite end of the licensing spectrum are permissive licences. These allow the adoption of any business model, including the creation of closed, proprietary derivatives from the openly provided shared code (see further Chapter 3). These licences rely on economic and community pressure to encourage contribution back to the project. Permissively licensed projects therefore require a well-defined community governance model.

In theory, the idea that all these community projects being shared through copyright licences means that the various projects can share their outputs and contribute to one another's code across the defined communities. Unfortunately, it is not quite as simple as this. Due to the incompatibility of some licences designed to prevent non-free derivatives of free software, such reuse is not always possible. In summary, permissively licensed code can be reused in code using a reciprocal (copyleft) licence but the reverse is not always true and this is also explored in more detail in Chapter 3. This situation is further complicated when we introduce the concept of 'partial' or 'weak' copyleft. Partial copyleft licences are ones that only demand reciprocal sharing of modifications to the free software but also permit embedding of this code in proprietary products. In some cases, partial copyleft free software can be included in permissively licensed open source software. It is out of scope for this chapter to go into detail about open source licence compatibility, and this is covered in Chapter 3. It is important to acknowledge the existence of this concern at this point, since licence choice clearly influences how and when communities can share their code and, to an extent, the nature of the communities themselves.

When sharing across projects is facilitated through the use of compatible licences we see immediate benefits in the code production cycle.[8] By sharing resources in the production of non-differentiating code, companies are able to reduce the cost and increase the quality of outputs. By 'non-differentiating software' we mean software that does not mark the participants as unique in the marketplace,

[8] Dirk Riehle, 'The Economic Motivation of Open Source Software: Stakeholder Perspectives' (April 2007) 40(4) *IEEE Computer* 25–32 , available from <http://dirkriehle.com/computer-science/research/2007/computer-2007-article.html> accessed 21 July 2022.

whether they provide software, services, or some other output produced through software use.

Regardless of licence choice not all Open Source software communities are open, collaborative communities. The licence guarantees that everyone has certain rights with respect to the use of the software code, but it says nothing about the development model adopted. In some cases the owners of the software may choose to maintain a development model in which only a very limited number of people are able to participate in the software development decision-making process. This may influence potential users' decisions to use the software or not, which in turn affects the likelihood of third parties contributing to the production of the software.

A combination of the development model adopted and the licence chosen for the software will influence the kind of community one can expect to find around a particular Open Source software project. This, in turn, influences the kind of revenue creation or cost-saving opportunities available to companies that are producing or consuming project components and which is explored more fully in Chapters 15 and 16 where we explore the economics and commercial models. We will return to these points later in this chapter, but first we will dig deeper into the political and ethical considerations that influence the decisions behind community structure and licence choice.

2.9 The Politics and Ethics of Open Source

So far, we have examined Open Source software in its role as a production technique in which an IP licensing model defines how the project outputs are shared outwardly and openly. We have indicated that this is only a part of the story and that there are also important political and ethical considerations to be taken into account. The term 'free software' may be used to refer to some of these issues. However, the term 'open source' does not necessarily exclude the same arguments since all open source software is also free software.

The term 'free software' was adopted by the FSF and pre-dates the term open source software. For some, it is the preferred term and they do not wish to associate themselves with the term 'open source' because it has become 'associated with a different approach, a different philosophy, different values, and even a different criterion for which licences are acceptable.'[9]

Free software must not be confused with 'freeware', which is software that can be acquired at no cost but for which source code is not available. Freeware provides none of the benefits of code-sharing that we see in free software. That is, whilst the cost of the software is zero it is not possible to adapt the software to suit one's

[9] <http://www.gnu.org/philosophy/free-software-for-freedom.html> accessed 21 July 2022.

specific needs. In addition, the lack of source code makes it impossible for users of that code to share their experience and thus reduce the cost of further development and software maintenance.

While freeware focuses on the lack of licence fee, free software is considered to be more of a social movement that adopts a specific IP licensing methodology, whereas open source is more of a software development movement using a broadly similar IP licensing model. For the free software movement non-free software is a social problem while for the open source movement it is a suboptimal solution. Whilst the two groups disagree on some basic principles, they do agree, in the main, on the practical recommendations they make. This section describes the differences between the two movements' basic principles.

It should also not be confused with Public Source or Shareware, which whilst having the source code publicly available lacks an approved licence granting one rights to use the code in a free or open source manner and is proprietary.

2.10 The Free Software Definition

The term 'free software' refers to software that respects users' freedom and community. Everyone has the freedom to run, copy, distribute, study, change, and improve the software. These freedoms ensure that users (collectively or individually) are able, if they so desire, to control the program and what it does for them. The FSF argues that when users are unable to control the program then the program controls them. As a result such 'non-free' software is sometimes seen as 'an instrument of unjust power'.[10] Free software is therefore a matter of liberty, not price: 'free' as in 'free speech', not as in 'free beer'.[11]

In order to establish whether or not software is free software, the FSF has defined four essential freedoms. To be free software the terms under which it is distributed and used must provide all four freedoms. These are:

- The freedom to run the program as you wish, for any purpose (freedom 0).
- The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1).
- The freedom to redistribute copies so you can help others (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3).

It is important to note that there is nothing in these four freedoms that would indicate that free software is non-commercial. In fact, the four freedoms ensure that

[10] <http://www.gnu.org/philosophy/free-sw.html> accessed 21 July 2022.

[11] <http://www.gnu.org/philosophy/free-sw.html> accessed 21 July 2022.

commercial use, development, and distribution are possible. Another important clarification is that the freedom to modify the software does not imply that third parties must accept your modifications. The value of any changes in the software is a subjective matter and the four freedoms do not seek to provide any guidance on the acceptance or otherwise of modifications. They only seek to ensure a users' right to make and redistribute modifications.

A free software licence may require a change of name and branding for a product that has been modified, as its trademark may not be available for use on modified software (see Chapter 9), but as long as these requirements are not onerous they are not considered to be a restriction on a users' right to modify (see further Chapter 24). This provision is to enable third parties to build value in their version of the software and thus generate revenue streams that will pay for further development of the software.

2.11 The Open Source Definition

We have seen that the four freedoms of free software focus on a perceived social need for software to be free. We have also seen that whilst the open source movement sees non-free software as suboptimal, its more pragmatic position serves to de-emphasise the ethical requirement for software freedom whilst recognising the importance of collaboration. Finally, we have seen that whilst the free software and open source software movements differ in motivation, they agree on most of the practical recommendations.

The equivalent of the FSF for the open source movement is the Open Source Initiative (OSI). The OSI provides the Open Source Definition (OSD)[12] which defines the ten considerations that an open source licence must address. These are:

1. **Free Redistribution**
   There can be no restrictions that prevent the software being distributed either alone or aggregated with other software. This includes no requirement for royalties or fees. However, as with free software, this does not mean that open source is non-commercial.
2. **Source Code**
   Software distributions must include source code in a form that is usable by a typical programer.
3. **Derived Works**
   The licence must allow derived works that can be distributed under the same terms as the original software. Note, this requirement does not extend

[12] <http://opensource.org/docs/osd> accessed 21 July 2022.

to other software distributed alongside open source software, only to derivatives of the open source software itself.

4. **Integrity of the Author's Source Code**
   If there is any restriction on the distribution of modified source code then it must allow distribution of 'patch' files to allow programers to reapply any modifications made. No restrictions on the distribution of binaries built from modified source, beyond requiring different branding, are allowed.
5. **No Discrimination Against Persons or Groups**
   The licence must be identical for all persons and groups.
6. **No Discrimination Against Fields of Endeavour**
   The licence must be identical for all types of use.
7. **Distribution of Licence**
   The rights assigned in the licence must apply to everyone who receives a copy of the program.
8. **Licence Must Not be Specific to a Product**
   The licence cannot depend on the software being distributed in a specific form or as part of a specific product.
9. **Licence Must Not Restrict Other Software**
   The licence must not affect other software distributed alongside the licensed software.
10. **Licence Must Be Technology Neutral**
    No provision of the licence may depend upon a specific technology or style of interface.

A careful comparison of the OSD and the Four Freedoms will show that they are compatible. Any software that is open source is also free, and vice versa. As we discussed earlier, the main difference between the two movements is philosophical and that the free software movement is driven by social need while the open source movement is driven by pragmatism. These different motivating factors lead to a possible division on the type of user that is best served by each model but which in current times is largely irrelevant.

2.12 The Open Source Initiative, a Pragmatic Community

The OSI is a non-profit corporation that was formed in 1998 to educate about and advocate for the benefits of open source software. It also seeks to build bridges among different constituencies in the open source community. The OSI defines open source as 'a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in.'[13]

The OSI chose the term 'open source' rather than 'free software' as it felt that the latter term had come to be associated with a philosophically and politically focused group. The OSI sought to focus more on pragmatism and the business case for the collaborative development of software. While the motivation for their advocacy was quite different to that of the FSF, the end result is largely the same practical behaviour: the development of free and open source software.

The OSI acts as a form of standards body, maintaining the OSD and a trademark that creates a nexus of trust around which developers, users, corporations, and governments can organise open source cooperation. In order to use the OSI trademark, software must be released under one of the licences that the OSI have reviewed and approved as being in conformity with the OSD.

The OSI's mission to define the conditions under which participants can openly collaborate on free and open source software is more difficult than the more tightly bounded mission of the FSF to build only free software or the ASF's mission to build only permissively licensed software (see ASF case study, section 2.14). The OSI seeks to be pragmatic and business-case driven, but it is hard to imagine a situation in which all-comers will converge on a single position.

The difficulty of reaching unanimous consensus across all parties led to the strange situation in which a foundation created to promote collaboration is itself a closed organisation. The OSI bylaws[14] do not allow a membership to be formed and thus all authority is vested in the Board of Directors. The Board consists of between five and twenty-one individuals, each of whom is elected by the existing Board of Directors. There are currently (2022) nine members of the Board of Directors. This arrangement allowed the OSI to complete the difficult task of defining the OSD and the associated licence approval process. However, this approach has limited the foundation's ability to have a significant impact beyond this initial work.

In 2008 an attempt was made to reform governance of the organisation. The OSI board invited fifty individuals to join a 'Chartered Members' group, forty-two of whom agreed. However, this group conducted its business on a private mailing list and its membership was never made public. The group made no visible progress on reformation of the OSI.

In 2012 an initiative was undertaken to transition towards a membership based governance structure. A free Affiliate Membership program has been introduced for 'government-recognised non-profit charitable and not-for-profit industry associations and academic institutions'. Individuals can also join as 'Individual Members' a small fee. A third phase was planned in which corporate members

[13] <http://opensource.org/> accessed 21 July 2022.

[14] <http://opensource.org/bylaws> accessed 21 July 2022.

would be invited to join, but has not (as of 2022) been implemented. At the time of writing (late 2022), none of these membership types provide any formal influence over the foundation as defined by the bylaws.

It is important to note that because the foundation does not generate outputs that contribute directly to the products and services provided by most commercial free and open source software companies, it has found it difficult to generate significant contributions in the form of volunteer energy. The OSI seeks to be more effective with the greater financial resources membership will make available. For example, the current Board of Directors expects the need for 'dedicated, long-term advocacy and organising' to require the provision of resource such as permanent staff and/or fellowship positions organising.[15]

2.13 Pragmatism versus Ethics

For the open source movement, the focus is on providing the maximum flexibility for producers of software. That is producers are free to do anything they want, including produce non-free software that incorporates open source software. The free software movement, on the other hand, seeks to protect the end-users' freedoms by ensuring all software supports the four freedoms.

Where producers choose to produce only free and open source software, users retain the four freedoms. However, where producers choose to include open source software in proprietary (non-free) software, those freedoms are, at least in part, lost.

The open source movement accepts that some producers of software will continue to produce proprietary software. This movement therefore seeks to ensure that all software producers can collaborate on open source software regardless of their chosen product licence strategy. Since a significant portion of the software industry is built on the capability of software producers to create false scarcity through the use of restrictive licensing models, this situation might be seen as a necessary compromise.

Despite the willingness of the open source movement to accept this compromise position it would be unreasonable to suggest that this compromise is a necessity. Ten years ago the author of this chapter for the first edition described the predominant model for software as being a proprietary controlled one. As a consequence of digital transformation, developer adoption and open source software now being found in up to 90 per cent of code bases, and up to 70 per cent of code[16] in 2022, this can be seen to be shifting. The pragmatism of the open source movement

[15] <http://opensource.org/members> accessed 21 July 2022.

[16] Tidelift 2020, <https://tidelift.com/subscription/2020-managed-open-source-survey> accessed 21 July 2022.

allows software producers to choose the point at which to draw the line between open source software and proprietary software.

It is this ability to define one's own boundaries that enhances the flexibility for the software producer when engaging with open source software projects. However, without the legal requirement of a reciprocal or copyleft licence to ensure project derivatives are also free software, the driver for development collaboration is less obvious. What is to prevent software producers benefiting from the open software without contributing to its development?

Whilst there is no financial cost attached to the unrestricted sharing and distribution of open source software there are costs to community partners that choose to not share their work in return. For example, a user of open source software may build a product that is in direct competition with other users of that software and this competition could be damaging to the business of those users that choose to collaborate openly. However, in this situation the non-collaborating partner will bear the full costs of maintenance of the software within their systems while the collaborating partners will share their costs. For successful re-users of open code, the costs associated with this maintenance will increase over time as the user continues to diverge from the publicly shared code. Consequently, over time, the economic pressures to collaborate increase.

Supporters of the more pragmatic open source movement point to these economic drivers for collaboration as evidence that, over time, more and more software will become a part of the open source project. Supporters of the free software movement acknowledge this but insist that the process should be accelerated by ensuring all distributed software (as opposed to modified software for personal use) must be made available under a free software licence.

2.14 The Apache Software Foundation

The ASF is a hugely influential organisation. It houses some of the most important Open Source software projects and has a long history of producing successful software. It is an example of an organisation that uses permissive licences to maximise the options for reuse, while a community-focused development model seeks to ensure that all participants have an equal influence on the project's strategy.

In this discussion we will see that this success is because the original creators worked hard to define a method of production that was as inclusive as possible. Unlike, for example, the FSF, the ASF focuses on the means of software production rather than the legal protection of software freedom. To many this is a suboptimal approach, but the model has been proven successful and repeatable in well over 356 Apache Projects Directory.

In February 1995 a small group of eight people created the Apache Group. These people had been independently maintaining the previously aborted, but public domain, NCSA web server. They had been sharing ideas and notes and decided that it was best to provide a level of legal protection and provide a limited amount of structure around their collaboration. The result was the Apache License V1.0, a permissive licence that allowed third parties to use the code in any way they desired. The organisational structure was informal but provided a means for coordinated decision-making within the group.

By 1999 the group had grown to twenty-one members and a multinational company wanted to use their software in a mainstream product. However, this company's lawyers were concerned about the lack of legal structure behind the licence. The solution was to create a US-based public charity (501(c)(3), in the parlance of the US Internal Revenue Codes). The mission of the foundation is to 'provide support for the Apache community of Open Source software projects. The Apache projects are characterised by a collaborative, consensus-based development process, an open and pragmatic software license, and a desire to create high quality software that leads the way in its field.' Perhaps more important than the mission is the tagline the ASF uses: 'not simply a group of projects sharing a server, we are a community of developers and users'.[17]

This emphasis on the 'community of developers and users' is present in the bylaws of the ASF and the licence used. The ASF operates with one simple goal: to ensure that a community of project developers can do what they do best—produce software for the benefit of all. The ASF exists only to provide the social, legal, and technical infrastructure to facilitate those developers.

The ASF places a very heavy emphasis on the social aspects of collaboration. All Apache projects adopt a development model that is often called the 'Apache Way'. This is a transparent, open, and meritocratic governance model that defines a small set of rules that all Apache projects observe. These rules ensure appropriate management of IP rights and community engagement. Those familiar with Apache projects will recognise that there is often more emphasis placed on community development than any other aspect of project management. This is possible because, for example, much of the legal overhead of managing an Apache project has been reduced by using a standard licence across all projects which significantly reduces the overhead in addressing legal issues in individual projects.

An important aspect of this community focus is a constant drive to ensure that all community members are seen as equal. For this reason the ASF does not pay for software development. All contributors to an Apache project are considered volunteers within the project. This means that there are no management structures within the projects and no individual's opinion is regarded as more important than any other. This is supported by a clearly defined meritocratic and consensus-based decision-making process that is surprisingly efficient in its application. This

[17] <http://www.apache.org/> accessed 21 July 2022.

equality among volunteers is extremely important to the ASF and is fiercely protected. It is this equality that allows anyone, regardless of the resources available to them, to contribute in a meaningful way whilst still being protected against the most serious potential abuses of their time.

The ASF uses economics to ensure contributions are made back to the Foundation wherever it is strategically appropriate to do so. That is, if a user of ASF software chooses to modify Apache code without contributing back they are introducing a maintenance overhead that other participants do not have. This means that competitors can enter the market and benefit from lower development and maintenance costs by actively engaging with the community. In many cases this reduction in costs can make them more competitive in the marketplace.

Traditionally Apache projects have focused on infrastructure components which are more easily reused in derivative products. However, in 2011 the foundation created its first significant end-user project in the form of Apache OpenOffice. This suite of productivity tools is intended to be installed and used as a single multifunction product. The success of this end-user product at the ASF is testament to the Apache model of software development working beyond infrastructure projects.

The ASF created a structure that has successfully and repeatedly produced Open Source that is used to deliver immense value to modern business. Its software is used in large Internet commerce sites, social networks, space-faring vehicles and control centres, government agencies, universities, schools, banks, and even children's toys. It is hard to imagine any computer that doesn't have Apache software embedded within it somewhere. All this is achieved through the provision of a genuinely neutral collaboration space, enabled by an Open Source licence, and a model of meritocratic, bottom-up decision-making that is both efficient and fair to all participants.

2.15 Governance of Open Source

The OSI describes open source as 'a development method for software that harnesses the power of distributed peer review and transparency of process'. Note that this definition does not reference licences. Nobody will deny that Open Source licences are vital. Once copyright was applied to computer software, conformance with the four freedoms and/or the OSI's OSD can only happen with the application of an appropriate and approved licences. However, there is more to Open Source than a licence.

To realise the full potential of Open Source (or its subset open source) as a development methodology we also need to consider 'governance models', that is how a project is managed. A clear governance model ensures that all contributors understand how to engage, what is expected of them, and what protections are provided for their status as a contributor. That is, it defines the rules of engagement and decision-making.

Open Source licences provide the legal framework within which parties collaborate. A governance model provides the social framework for that collaboration. A combination of Open Source licence and a transparent governance are what makes Open Source successful. So what is a good governance model?

2.16 People versus Process

An Open Source governance model need not be, indeed should not be, a complex document that attempts to cover every possible circumstance. The governance evolves over time to be the collection of documentation that enables the community to get work done as members come and go. It may start with a few processes and evolve as the project evolves. It captures the processes in such a way as to transmit the culture to new members. It prevents a project from becoming stuck because the institutional knowledge resides in a few members' heads. It is guidance documentation covering the most common community processes, nothing more. Examples of such processes may include:

- How to file a good bug.
- How to suggest a new feature.
- How to propose a patch or pull request.
- How are features for a new release of the software project decided.
- How are new releases of the software project built.

The governance should recognise that written rules can be both empowering and constraining. Rules make a process predictable and repeatable, but they can also make a community resistant to change or even blind to the need for change. The goal is to create an environment in which people feel comfortable engaging with the project on a long-term basis. The governance should hopefully account for its own evolution.

In reality, most software developers just want to be able to get work done in an efficient way. This is especially true in a community-led Open Source project. A good governance model is therefore about enabling flexibility, empowering individuals to lead on specific activities, and preventing (occasionally resolving) conflict.

However, this in turn can present a problem. Many people find working in a self-directed, bottom-up, collaborative environment challenging. This is where leadership comes in. In an Open Source project leadership is not about directing but instead is about getting results and empowering others.

A common concern about community-led Open Source projects is that they will quickly descend into anarchy because they adopt a bottom-up, leaderless approach to coordination. Finding the right balance between bottom-up anarchy and top-down leadership is hard. This is where the governance model applies. It provides the social scaffolding for collaboration. It empowers individuals who just want to get things done and it provides mechanisms by which community deadlocks can be broken.

An Open Source licence is only a small part of this governance model. As we discussed earlier, some licences legally enforce a sharing of code modifications. Others depend upon economic and social pressures. Choosing the right licence and the right style of project governance is critical to the success of an Open Source project. It is important to understand some of the choices a project must make about its social governance.

Sections 2.17 and 2.18 outline two examples of common governance models in communities. These two approaches appear to be diametrically opposed. The first is the benevolent dictator model, where a single individual has absolute authority. The second is the meritocratic model, where valuable contributions are rewarded with collective leadership authority. Once you understand both of these 'extremes' you'll see that, in practice, they are not as dissimilar as they first seem.

2.17 The Benevolent Dictator Governance Model

A benevolent dictator (BD) is an individual who has complete control over the decision-making process in an Open Source project. Linus Torvalds[18] is perhaps the most well-known benevolent dictator. Being a BD is not an easy job. It requires diplomacy and community building skills, in-depth technical knowledge of all aspects of the project, and exceptional levels of commitment and dedication. However, as Linus' Linux Kernel[19] project illustrates, it can be very effective.

The BD model relies heavily on the fact that an Open Source licence allows anyone to take the code and spin up their own project or fork. This means that although the leader has full control over their project, they must still work to ensure that community needs are met. Failure to do so will result in a splintering of the community as objectors set up their own projects based on the same code and the community walk off with their code base. A BD who wishes to create a vibrant community project must therefore seek to ensure each decision is both understood and supported by as many community members as possible. Consequently, diplomacy, mediation, and clarity are just three of the softer skills that a BD needs.

Although BDs are usually highly skilled from a technical perspective, they are unlikely to be the best person to make every technical decision. A good BD recognises this and seeks to enable the community to collectively make decisions under

[18] <http://en.wikipedia.org/wiki/Linus_Torvalds> accessed 21 July 2022.

[19] <http://www.kernel.org/> accessed 21 July 2022.

the guidance of the most skilled members. When the community is unable to reach consensus, the BD will intervene by making what they believe to be the most appropriate decision. In this way a BD seeks to prevent the project from becoming paralysed by indecision.

It is therefore the BD's job to resolve disputes within the community and to ensure that the project is able to progress in a coordinated way. In turn, it is the community's job to guide the decisions of the BD through active engagement and contribution. Consequently, the BD model can scale because in the majority of cases, the BD is not needed to allow the community to progress.

Typically, the BD is self-appointed. They will usually be the originator of the project or their appointed successor. In many ways, the role of the BD is less about dictatorship and more about leadership and diplomacy. The key is to ensure that, as the project expands, the right people (those who concur with the BD's vision) are recognised as community leaders. A BD who does not have or does not maintain a following within a project will quickly find the project has forked and themselves usurped.

2.18 The Meritocratic Governance Model

In centralised models of governance, the gating of contributions through a single individual becomes a bottleneck. A meritocracy recognises this and provides a defined mechanism by which individuals can earn direct influence over the project. This process is quite different to approaches within which employment status, experience, or financial contributions might earn 'power'. In the meritocratic model, anyone contributing in any positive way earns equal authority. This process of empowering those who contribute scales very well. Furthermore, it minimises friction because it recognises power and influence as scarce resources. Newcomers are seen as volunteers that want to help, rather than people that want to grab a share of that scarce resource. A true meritocracy lacks artificial filters for contributions that are commonly found in other models. An example of such artificial filters is the ability to buy influence with cash rather than technical contribution. This lack of artificial filter ensures the broadest possible range of contributors who are aligned to a common goal. When managed well this process creates an environment in which everyone, regardless of their relationships outside a project, can collaborate. However, because there is no defined leader there needs to be a clear set of rules by which the community operates. Failure to provide clear rules of engagement usually results in a model that looks more like a top-down leadership model than a bottom-up community model.

A meritocracy is typically leaderless; that is, whoever is best equipped to lead in any specific situation will be the leader for that situation only. People lead through action, not authority; they don't have any more authority than any other participant. This often causes newcomers to meritocracy to assume that a project will inevitably grind to a halt since it will be unable to make decisions. In a healthy meritocracy it is possible for those with less experience to drive a given objective forwards through action, but since all work is in the open, those with more experience (but less time) will provide feedback. Where that feedback is seen as appropriate it will be recognised as meritorious and included in the final outputs. Fortunately, in software development, the vast majority of decisions are easily reversible. So long as missteps are identified early, they can be reversed with minimal negative impact. Consequently, most decisions in a software project are made through a process called 'lazy consensus' where the community lazily assumes that anyone with sufficient merit to take action is going to do so with good intentions. The community reviews all actions quickly and, if necessary, raise, discuss, and act upon any objections. In the unlikely event that consensus cannot be reached, a conflict resolution process is enacted.

Potentially controversial actions may be brought to the community's attention for feedback and approval prior to work being carried out. This can reduce the number of 'roll-backs' necessary since consensus is sought before work commences. Since those with merit have already demonstrated a sensitivity to when this is necessary there need not be hard rules in place to manage this, all that is needed is full transparency on all actions and their motivation.

2.19 Implications of Licence Choice and IP Management on Governance Models

The BD presents an organisationally simple model that can work extremely well, but only if the right leader can be found. The meritocratic model, on the other hand, does not depend on the availability of a single individual to act as project leader; however, it brings with it a more complex social structure that requires more engaged governance processes.

A potential downside of the benevolent dictator model is that it requires the community to trust a single individual completely, both today and in an undefined future. For many people, this dependence on a central figure puts the project at risk since individual circumstances change, as do employers' objectives. In a meritocratic model, contributors need not put their trust in a single individual but they must, at the very least, actively monitor the community to ensure it remains aligned to their own goals. The use of an Open Source licence, which allows any community member to 'fork' the project and make it their own, goes some way towards protecting the community. However, the choice of licence and the treatment of contributed IP can have a significant effect on a community's confidence in this model.

As an example of this interplay between licence choice, IP management, and project governance, consider the fact that a reciprocal licence coupled with copyright assignment of all contributions centralises legal control. The concern here is that it is not difficult to acquire full control of the Open Source project. This is especially true when a BD can exercise complete control over the project. With full control of both the legal and community aspects of the project it would be possible to act in ways counter to the community's interests. At its extreme this would mean the copyright holder can make all future development work proprietary while third-party contributors would still be bound by the original reciprocal Open Source licence. Thus there can be significant potential benefit to the 'owner' of a centralised, copyleft project.

In order to minimise the risk, one could seek to manage copyright in third-party contributions differently. For example, rather than centralising ownership in an organisation that can be acquired, one could use a suitable non-profit vehicle. This can ensure the safe-keeping of all contributions for the community because, for example, it would not be possible to purchase the assets of the non-profit copyright owner. Alternatively, one could avoid centralising copyright in the first place. This is achieved by only requiring contributors to grant a licence to reuse contributions in Open Source software, as opposed to assigning copyright to a centralised owner. In this instance re-licensing of the code would require the permission of all contributors.

It is also possible to minimise the impact of a community leader closing future developments by using a permissive licence rather than a reciprocal one. This does not prevent the creation of proprietary derivatives, but it does mean that all community members have equal rights to do so. However, in this instance we have moved the problem from being a legal one to becoming a social one. As we have seen in earlier sections, this tension between the pragmatic and ethical case for open source versus free software requires some careful consideration.

The matrix following presents an overview of the kinds of project that are created by various combinations of licence and copyright ownership. It should be noted that there is a third licence type, known as 'partial copyleft' which is not displayed in this model. A common partial copyleft licence is the GNU Lesser Public License (LGPL). This licence requires any derivatives of the code to be released under the LGPL, that is it is reciprocal. However, unlike a full copyleft licence such as the General Public License (GPL), LGPL code can be included in unmodified form in proprietary products. We have not included discussion of this licence type here in order to simplify this chapter. We feel justified in this decision since the FSF does not recommend the use of partial copyleft licences in most circumstances[20]

[20] <http://www.gnu.org/philosophy/why-not-lgpl.html> accessed 21 July 2022.

(despite the fact that the FSF is the author of the LGPL). Licensing is discussed in depth at Chapter 3 and contribution at Chapter 4.

		Copyright Ownership Model	
		Centralised	**Distributed**
Licence type	**Reciprocal**	**Economic Community**	
		All licensees have the same rights under a reciprocal licence and thus the status of copyright owner has no special bearing. All licensees are free to share modifications or to withhold them.	
	Copyleft	**Owned community**	**Enforced community**
		Under a reciprocal licence all licensees have the same rights. All modifications affected by the licence must be made available under the same licence. However, the centralisation of copyright means that the copyright holder is not bound by this same requirement and may choose to issue modifications under a different licence.	Under a reciprocal licence all licensees have the same rights. All modifications made by licensees must be made available under the same licence. Where copyright in each contribution is owned by individual contributors no single entity is entitled to withhold modifications affected by the licence.

It is outside the scope of this chapter to explore the many interactions between licence choice and project governance. Our intention here is to simply highlight the interrelationships between the two by providing a few illustrative examples. As you continue to read this book you will come to identify many more cross-dependencies between the legal and social models of Open Source software development.

2.20 The Rise of Codes of Conducts

There is a governance trend in the past decade that is on the rise and ubiquitous, and that is for Open Source projects to have a code of conduct or respect. This most likely comes about as several industry trends are reflected in Open Source communities.

Over the past couple of decades, we have seen a rise in companies having such codes of conducts. They are sometimes dressed up as standards of business conduct or business ethics. They are sometimes based around a theme such as 'trust' or 'integrity'. Good ones always set expectations for how the relationship with the company as an employee, customer, or partner should be viewed.

Over several decades, a recognition and a growing number of studies across a collection of industries have demonstrated that diverse teams build better value for companies' long term and have measurable impact on the bottom line. While some of the early studies may have been focused narrowly on gender, more recent studies have demonstrated such bottom-line impact from diverse teams in more nuanced ways. The software development profession is one such male-dominated profession and therefore the earlier period of software collaboration was equally imbalanced.

As these communities were online communities, communications were typically email distribution lists, online forums, and chatrooms. Without a company responsible directly for such communications channels, and with many channels allowing for the complete anonymity of the participants, some conversations could degenerate, becoming aggressive to the point of toxicity. Even in situations where participants were well known to one another, there are many documented incidents where conversational styles could be direct or blunt to the point of rudeness. Such bad behaviour can have a lasting effect on a project. One documented case showed that while the worst participants were finally driven from the community, the general participation in the community overall had dropped by 20 per cent and never recovered.[21]

Conferences supporting Open Source licensed projects became some of the first places to support a code of conduct as organisers worked to encourage more diverse participation in the conference itself. That model seems to have carried into the projects themselves. Some of the older projects recognised the need for more diverse participation from contributors and by extension their maintainers as they worked to refresh the original group that was approaching retirement age. Modern well-run projects now typically support a code of conduct. Non-profits supporting projects generally encourage their projects to have such a code of conduct.

A generally acknowledged well-written code of conduct is the Contributor Covenant. Started in 2014, a group has been maintaining the document and releasing new versions.[22] It generally reflects new developments in a broad collection

[21] D Berholz, 'Assholes are Ruining Your Project', available at <https://redmonk.com/tv/2012/04/06/assholes-are-ruining-your-project-donnie-berkholz-redmonk/> accessed 21 July 2022.

[22] <https://www.contributor-covenant.org/> accessed 21 July 2022.

of Open Source project communities and is run as an Open Source project in itself (using a Creative Commons licence as it is document based).

As more projects adopt such codes of conduct, best practices are also growing. One interesting case in 2018 demonstrated in a large community with a code of conduct that they didn't have a policy for what to do when there was a reported code of conduct violation. The principle organisers in that project immediately spent a few days adding such a policy to respond quickly and decisively to the situation in front of them.

2.21 The Business of Open Source

We have considered the models of collaborative community building and liberally licensed sharing with respect to Open Source project communities and governance. A detailed consideration of these in a commercial context is made in Chapter 16. There are two models for utilisation of Open Source software at a high level: a company can consume project components and Open Source, and a company can produce project components and Open Source. It is easy to see how a company with a broad software portfolio could do both and many companies move from being consumers to producers or contributors over time as is discussed further in Chapter 19.

We have already seen the easy economics behind consuming such projects in the GCC example earlier in the chapter using our borrow-and-share concept. It is a source of rapid value capture and defrays engineering development and maintenance costs over time for the component. It is also a direct source of innovation. Perhaps the best example of such value capture is Red Hat Inc. and the Red Hat Enterprise Linux product, as well as their own Fedora community distribution project. They invest modestly in the Linux kernel community[23] but reap enormous value.

An easy modern example of a vibrant community evolving around a project with companies pulling the project component into company products and services for customers, is the Kubernetes project. This Open Source project was started by Google, and Google invested to build a vibrant, diverse community around the project. The project's IP is held neutrally in a non-profit, the Cloud Native Computing Foundation (CNCF), giving other partners and members confidence that the ownership and management of the message is a level playing field. Many companies then pull from the Kubernetes project components to build and offer their own Kubernetes-based orchestration services, plumbing in their own network and storage drivers, management portals, and billing services. CNCF and

[23] 2017 State of Linux Kernel Development, <https://go.pardot.com/l/6342/2017-10-24/3xr3f2/6342/188781/Publication_LinuxKernelReport_2017.pdf> accessed 21 July 2022, p 14.

Kubernetes are amongst the greatest success stories of Open Source and underpin much of cloud computing today.

The products around Kubernetes have a valuation today in the trillions of US dollars and investment in the tens of billions of dollars based on the CNCF cloud landscape. In the way that Linux sits at the core of a large proportion of today's server market, so Kubernetes sits at the core of much of the cloud infrastructure, creating not only a community but also an ecosystem.

This distinction between project and product is key. When he was CEO for MySQL Inc., Mårten Mickos offered the observation:

The community has time and no money; customers have money and no time.[24] Community projects are not customer products. A company working with open source component projects needs to know when it is engaging community members, and when its talking to customers. These are distinct conversations with different goals and metrics for success.

When a company or public body produces its own open source components to share, one needs to do so thoughtfully. At a high level, a software company, one selling solutions to customers largely based on building software, builds software of three types:

- Context: Software tools used to build customer-facing products.
- Complement value add: Software that supports core products and services, creating a richer customer experience, and making the products and services sticky.
- Core value proposition: The software at the heart of the company's solution to customers, sometimes called a single vendor product.

Investing in building a modest community around 'Context' projects provides easy, modest returns. A good example is Netflix and the Spinnaker CI/CD (Continuous Integration/Continuous Development) pipeline project. There is nothing about Spinnaker that is core to the delivery of streaming video to all of its customers' devices. It is enabling software. Sharing it and others use of it creates a validation of the toolset and approach to the problem through wider use; innovation capture as others stress-test the Spinnaker project in new environments and contribute back; and recruiting benefits both as culture advertisement and as screening and self-selection for the problem space. There is little risk or liability management needed that can't be absorbed by the company and so the value of building a community is plain to see.

A good example of a 'Complement' project is the Microsoft Azure CLI. It is an Open Source project from Microsoft that provides a command-line interface to

[24] Keynote at the Open Source Business Conference, 2006, San Francisco.

Azure cloud services instead of the main Azure management portal. Investing in building community allows for deep partner engagement and customer co-creation, where partners and customers commit with Microsoft to the core technologies. You can see this in other larger-scale projects from companies: Microsoft with VS Code and .NET Core, and Red Hat with OpenShift.

The investment made by the parent company in building community needs to be long-term and should be scaled up to meet expected returns and anchored on core products and services. Individual developers may not buy software in today's world, but operational deployment in production data centres depends upon dependable, supported products being used.

Open sourcing by publishing software under an OSI License (and possibly building community) around a company's core value to customers requires deep practice in Open Source and experience to be able to separate the 'software' that makes up the customer solution from the customers' perceived value of the solution. Red Hat did this exceptionally well over a thirty-year period. After ten years, Red Hat pivoted from competing to be the best Linux distribution (distro) using Red Hat Linux, to becoming cheap-UNIX-on-Intel-servers in the data centre with Red Hat Enterprise Linux (RHEL) reacting to the shift in the marketplace and the move to cloud or platform as opposed to on premises (or on-prem), as their customers began scaling up their data centres. There was no confusion around Linux being a liberally licensed project and the value of RHEL to paying customers in this new environment.

Likewise, Red Hat carefully managed its trademarks and brands. It chose the route of distinct enterprise and community brands to ensure that customers clearly understood what was delivered as a commercial product as opposed to the community version. Using the company brand on a community project instead of a customer product may confuse customers and community members further, although some companies creating Open Source work with a single brand across both and deal with a difficult trademark dance and associated brand value issues, as discussed in Chapter 9 on trademarks and Chapter 16 on business models.

A quotation from Theodore Levitt captures the challenge best: 'People don't want to buy a quarter-inch drill. They want a quarter-inch hole!' Ensuring that you understand the problem you solve for your customers and can explain it in terms other than the software itself means that you might be able to have healthy Open Source project components in the heart of your core value proposition to customers.

Managing healthy project communities created in a company can be a challenge. Context projects easily fit into an engineering department. Complement projects need product alignment. Core projects are the company's bread-and-butter. Companies often make the mistake with Complement and Core projects in believing that there will be a conversion ratio of some kind from community project member to paid product customer.

This confusion may arise from a misunderstanding of Geoffrey Moore's chasm metaphor.[25] His model proposed that there is a gap between the cutting-edge first customers for a product and the more risk-averse early majority on the bell curve of technology adoption, and he proposed ways to cross this 'chasm'. There has been a collection of companies attempting to attract early adopters by publishing the core of their solution as an Open Source project.

The problem with early *project* adopters is they are not customers. They are giving the company valuable time (but per Mickos, 'have no money'). They are validating the project technology without valuing the customer-facing solution. Indeed, early adopting, sophisticated community members may be happy to solve their problems using their time but never get close to becoming customers. There is also the risk that they further confuse a company's actual customers and partners. Trying to sell to these community members only serves to infuriate them. Of course, some early customers may appear in the project community to validate the technology (as project) and may solve their problem, then be interested in buying the product, but if a software company doesn't know how to create the different on-ramps to separate community from customers, it will create headaches for everyone.

Open Source projects that sit on company core-value propositions frequently have awkward community dynamics. All the discussions of community building and governance earlier in the chapter assumed honest intent to build a community of equal participants, even when organised around a benevolent dictatorship. A company with a core-value project can have difficulty separating its views of product features and value from project functionality. These companies have primary ownership of the assets and they need to drive business.

- If the company is trying to run the project around the basic functionality but sell 'enterprise' features (sometimes referred to as an Open Core business model), they often run into potential partners and sophisticated end-users trying to contribute the enterprise features into the project rather than becoming full partners or paying customers.
- Partners want relationships based on co-investment, co-marketing, and co-selling. They generally are not interested in simply contributing to the OSI-licensed project to the primary benefit of the company holding the project.
- Competitors and potential partners can use the OSI-licensed project outright to support their customers without engaging in any partner relationship of mutual growth and benefit with the project originator.

[25] G Moore, *Crossing the Chasm*, 3rd edn (New York, NY: Harper Business, 1995).

- Competitors and potential partners can further begin differentiating with their own features in their own products and services depending on the licensing.
- Venture Capital (VC) funders whose core interest is return on revenue may also drive the company further down these roads in respect of Complement and Core products and issues such as the community reaction and forking as a response to this, as was seen in early 2021 with Elastic, become an inevitable consequence of these.

If the project-owning company attempts to curtail the community discussion too harshly, and there is no real community engagement in the project, then the company risks their message to customers that the solution provides the customer the benefits of an open code base and open innovation as opposed to Open Source, in a shared source or public source way.

If the company has invested in building a vibrant community, it may upset the very community members that are its strongest advocates and if it pushes them too far, the company risks a fork in its community members or product. There have been few occasions in the past twenty-five years where a community around a project has been stressed to the point of fracture, but it happens if the conditions are right[26] and the 2021 Elastic situation was one such example.

The tensions created in some businesses trying to use Open Source projects as their core offering have led in recent years to the companies having to re-licence the software under non-OSI licences, and led their investors to call in frustration for a re-evaluation of the OSD which has been met by the OSI with a very firm 'no', based on the need for continued certainty in this space.

A third way using shared source or public source are lesser-known alternatives to Open Source which do not, due to their nature, have an Open Source licence but a proprietary licence for the code and also are unlikely to build community or receive the benefits that it brings. Although there has been some drive to create movements and traction around these concepts over several years and again in 2021 following on from Elastic's move to the proprietary SSPL licence, no real traction has ever occurred, and this is probably due to the lack of developer community engagement that these alternatives have, and it seems very unlikely that these will really gain traction.

Companies creating Open Source project components in context and complement spaces are powerful tools for engineering, partner, and customer co-creation. Companies publishing the core of their value proposition as an Open Source project need careful messaging and planning, as is discussed in some detail in Chapter 16.

[26] Memorable examples of community forks include: GCC vs EGCS, OpenOffice vs LibreOffice, Jenkins vs Hudson, Compiere vs Adempiere, ownCloud vs Nextcloud.

All of this really boils down to the simple view that Open Source projects are powerful tools to capture value and innovation, and community engagement is a great way to protect and grow that investment, but at the end of the day, a software company still needs to run the business. This can be seen in the simple comparisons in public company filings.

Whilst Red Hat reported a gross profit margin of 84.91 per cent in 2019; Microsoft, a leading provider of proprietary solutions, reported gross profit margins of 64.4 per cent as on 30 June 2022. Apple, a leading supplier of hardware supported by proprietary software, reported 43.31 per cent as on 30 June 2022, and Google, a software-based service provider, reported 56.75 per cent margins as on 30 June 2022.[27] All of these companies use and contribute to Open Source software. But only one of them can be considered an Open Source software business in the sense that all their products are available under an Open Source software licence: Red Hat. So, whilst Open Source as a central concept in a business plan must be carefully thought through, it is clearly not without merit.

2.22 Open Source Non-Profits

We have discussed a number of non-profits through this chapter. These organisations have provided valuable structure to the work of successful Open Source project communities, and have enabled a great deal of success for subsequent uses of the project components in businesses. We have met each of them from a historical perspective and each was an experiment conducted at the time to address a particular challenge. It is important to take a look at them now collectively as they are becoming more common, and understanding their structure and use will hopefully create better outcomes.

Successful Open Source project communities do the work to build on-ramps to attract users, developers (many of whom will selfishly experiment rather than contribute to the open source community), and hopefully encourage developers to become contributors. Contributions are the life blood of successful projects.

There comes a time, however, when the project reaches a point in its growth where companies want to become users and contributors to the project, and a company's risk management profile is often higher than casual participants in an Open Source licensed project. At the same time, as a project grows, so does the liability of the project maintainers. These two constraints limit a project's ability to grow. Non-profit corporate legal structures can alleviate both problems.

[27] All gross margin statistics from <http://ycharts.com> accessed 21 July 2022.

- The corporate framework removes personal liability from maintainers and IP holders.
- It organises IP management around the copyright and trademarks in a neutral and improved manner.

Solving for these two problems enables companies with higher risk management needs to consume project components into customer-facing products and contribute back to those projects. This then expands the user and contributor pool and by extension (hopefully) the maintainer pool. In extending the contributor pool, it can do so with full-time employees, depending on the nature of the project and its use in products.

This doesn't create a market but it enables broader use and enables more participants in a project community. The non-profit creates safety around the code base which allows a market to form. In market design terms, a non-profit can make the market bigger and safer.

Non-profit organisations hold the bank account, can sign contracts (for conferences, etc), can provide standardised infrastructure services, and can also provide messaging and educational services. From a market design perspective, it allows information to flow freely to all project participants (users/contributors/maintainers).

A class of non-profits have been structured as member organisations over the past twenty years, and this allows companies to invest collectively in supporting Open Source licensed projects with services for growth, and providing a common messaging platform, while providing anti-trust protection to the members. These are explored in more detail in Chapter 18.

All of this sounds like a great boost to Open Source projects but there are considerations to be managed:

- Work needs to be done. In the same way that a project is only as successful as the participants willing to work on it, so to with non-profit organisations.
- Creating the legal framework and basic accounting services costs money. A group of funders who care needs to invest to create and maintain the non-profit.
- Growth depends on healthy projects. Non-profits can't make unhealthy community cultures healthy. Indeed, non-profits amplify the existing culture.
- If a company-owned Open Source project is brought into a non-profit structure and there is already tension in how the community is managed with respect to project and product, the non-profit can't solve for the culture mismatch

Indeed, you can end up with a collection of struggling projects around a message. Non-profit members often create a set of messages around the projects within the organisation. This messaging platform is important in and of itself and can become the centre of gravity for the non-profit rather than supporting the proprietary projects themselves.

Non-profits are important organisations in the Open Source jigsaw puzzle. But it is important to understand their uses and limitations and these are discussed further at Chapter 18.

2.23 Conclusion

Collaboration of the type we find in Open Source software is not a new concept. We have been building collaboration for individual and group benefit for at least as long as we have been defending our territory. Proprietary software teams may choose to collaborate beyond their organisational borders but here, sharing is the exception rather than the rule.

Open Source software communities and business teams, on the other hand, make sharing the default position. They rely upon the fact that the reproduction of completed software components does not consume a scarce resource. Furthermore, company participants in Open Source communities recognise that because large portions of their products can be standardised and thus shared with collaborators at minimal cost, there is an opportunity to reduce the initial cost of production and ongoing costs of maintenance for their businesses.

Fundamental to the success of this model of software production is the adoption of an Open Source licence. It is the licence that protects each participant from exploitation. The licence seeks to ensure that all contributors remain on equal terms. For this to work one needs to consider carefully both the licence chosen and the processes adopted to allow the software to be released under this licence. Chapter 3 will explore, in detail, the role of Open Source licences.

PART 1

INTELLECTUAL PROPERTY, CORPORATE, AND GOVERNANCE

3
Copyright, Contract, and Licensing in Open Source

P McCoy Smith

3.1 Copyright and Software

3.1.1 The history of software and copyright

Although the history of manmade computing devices reaches into antiquity,[1] it was not until the development, beginning in the 1950s, of high-level programing languages for authoring and editing computer instructions,[2] together with existing systems for transcribing and loading those instructions into a computing device,

[1] Jo Marchant, 'Decoding the Antikythera Mechanism, the First Computer' *Smithsonian Magazine* (February 2015) <https://www.smithsonianmag.com/history/decoding-antikythera-mechanism-first-computer-180953979/> accessed 11 November 2019.

[2] FORTRAN (a portmanteau of 'formula translation'), first used in 1954, is generally considered to be the first high-level programing language, and remained the dominant programing language for scientific and mathematical computing well into the late twentieth century. See 'Fortran', Techopedia, <https://www.techopedia.com/definition/24111/fortran> accessed 21 December 2016. In fact, several important benchmarks for measuring and comparing performance in supercomputing use FORTRAN, given its frequent use in highly complex scientific and mathematical calculations. See SPEC CPU®2017 Floating Point <http://www.spec.org/cpu2017/Docs/#benchmarks> accessed 13 April 2022.

P McCoy Smith, *Copyright, Contract, and Licensing in Open Source* In: *Open Source Law, Policy and Practice*.
Edited by: Amanda Brock, Oxford University Press. © P McCoy Smith 2022. DOI: 10.1093/oso/9780198862345.003.0003

that software began to resemble those things—such as literary or other artistic works—for which Intellectual Property (IP) rights had previously been extended. The ability to author, adapt, reproduce, and systematically load programing instructions into a computing device using a combination of a writable medium, and a high-level programing language—understandable to a multitude of human programers and (after compilation) computing machines—first began to cause computer scientists, lawyers, and legislators to contemplate forms of IP protection that might be used for programs created using a combination of these mechanisms.

In the 1960s, the potential for using copyright as a mechanism for securing exclusive rights to computer code started to emerge, and test cases were attempted in the US to establish the application of copyright to software.[3] Thereafter, the Commission on New Technological Uses of Copyrighted Works (CONTU) was established in the US in order to study what, if any, IP protection might be appropriate for 'new' technologies like software.[4]

CONTU's eventual report, issued in 1978, recommended that copyright protection should be available for computer programs composed of 'a set of statements or instructions to be used *directly or indirectly* in a computer in order to bring about a certain result'.[5] This recommendation was not without dissent,[6] as some committee members felt that computer software was primarily or exclusively functional and therefore not an appropriate target for copyright protection, but instead should only be protectable by patent. The US Congress amended the US Copyright Act in 1980 to specify that computer software was within its scope.

The change made in US copyright law to reflect the recognition that software fell within its ambit were later included in law in the UK[7] and the EU[8] to give similar recognition.

Modern computing devices are typically configured using, and have data input in, information in a format often referred to as 'binaries' or 'executables',[9] instantiated as a lengthy string of '1' and '0' values. In the early days of the application of copyright to software, there remained the question of whether that

[3] See George D Carey, 'Copyright Registration and Computer Programs' (1964) 11 *Bulletin of the Copyright Society of the United States of America* 362, at 363; General Atomic Division of General Dynamics, 'Gaze-2, A One-Dimensional, Multigroup, Neutron Diffusion Theory Code for the IBM-7090', US Copyright Registration No. A607663 (registered 1 January 1963) (a FORTRAN program, considered to be the first registered copyright on software in the US).

[4] National Commission on New Technology Uses of Copyrighted Works, 'Final Report' (31 July 1978) (CONTU Report).

[5] CONTU Report, see note 4, Ch. 3.

[6] CONTU Report, see note 4, Ch. 3, Concurrence of Commissioner Nimmer (arguing that only certain types of computer programs should be afforded copyright protection) *and* Dissent of Commissioner Hersey (arguing that copyright protections should not be afforded to computer programs at all).

[7] See UK Statutory Instruments 1992 No. 3233 The Copyright (Computer Programs) Regulations.

[8] See Council Directive on the Legal Protection of Computer Programs, 34 OJ EUR. Comm. Mr. (No. L 122) 42 (1991).

[9] See *Encyclopaedia Britannica*, 'Binary Code' (19 January 2020) <https://www.britannica.com/technology/binary-code> accessed 10 February 2020; *PC Magazine Encyclopedia*, 'Executable Code' <https://www.pcmag.com/encyclopedia/term/executable-code> accessed 10 February 2020.

form of software was eligible for copyright protection, given the seemingly pure functionality of the way that code was used and its general incomprehensibility in that format to even skilled programers. A similar issue had confronted courts in both the US and the UK in the early twentieth century, in cases involving 'piano rolls'.[10] In both countries, the courts determined that this format was *not* an infringement of the copyright in the underlying musical composition: '[T]o play an instrument from a sheet of music which appears to the eye is one thing; to play an instrument with a perforated sheet which itself forms part of the mechanism which produces the music is quite another thing',[11] and 'These perforated rolls are parts of a machine which, when duly applied and properly operated in connection with the mechanism to which they are adapted, produce musical tones in harmonious combination. But we cannot think that they are copies within the meaning of the copyright act.'[12] Although these decisions would—if still applicable today—provide a sound basis for denying copyright rights to (or at least, disallowing infringement claims by source code authors against) executable code created using copyrightable source code, subsequent legislative developments reversed the outcome of those decisions.[13] The piano roll decisions, and subsequent legislative changes to reverse them and later to bring computer software within the purview of copyright, were influential—if not dispositive—on the question of whether executable code fell within the protection of copyright when challenges to that proposition were made in the early 1980s.[14] As a result, it is without question that a programer's copyright rights subsist in any instantiation of their authored code—from the form in which the author originally wrote it (in most cases, source code), to any subsequent human or machine translation of it into any intermediate or final format (e.g. object code or executables/binaries), that is understandable by a computing device and upon which it may act, as long as the work otherwise falls within the boundaries of copyright and outside of any exceptions thereto.

[10] A piano roll was a long roll of paper with punched holes which, when fed into a specially adapted piano (called a 'player piano') would convert the information on that roll into played music. See *Encyclopaedia Britannica*, 'Player Piano' (9 September 2019) <https://www.britannica.com/art/player-piano> accessed 10 February 2020.

[11] *Boosey v Whight*, 1900 1 Ch. 122, 81 LTNS 265.

[12] *White-Smith Music Publishing Co. v Apollo Co.*, 209 US 1 (1908).

[13] See Kal Raustiala and Christopher Jon Sprigman, 'Scales of justice: How a terrible Supreme Court decision about player pianos made the cover song what it is today' *Slate* (12 May 2014) <https://slate.com/technology/2014/05/white-smith-music-case-a-terrible-1908-supreme-court-decision-on-player-pianos.html> accessed 13 April 2022; UK Public General Acts 1911 c. 46, Part I, Section 1(2)(d) ('in the case of a literary, dramatic, or musical work, to make any record, perforated roll, … *or other contriavance by means of which the work may be mechanically performed or delivered*'); 17 USC § 102 (2018) ('Copyright protection subsists … in original works of authorship fixed in any tangible medium of expression, now known or later developed, from which they can be perceived, reproduced, or otherwise communicated, either directly *or with the aid of a machine or device*.').

[14] See *Apple Computer, Inc. v Franklin Computer Corp.*, 714 F.2d 1240 (3d Cir. 1983), *cert. dism'd by stip.*, 464 US 1033 (1984).

3.1.2 The author's rights in software copyright

Copyright defines a set of legal rights which are granted—via national law, but in most jurisdictions consistent with the Berne Convention for the Protection of Literary and Artistic Works[15]—to the authors of a work eligible for protection. As discussed earlier, although the question of whether software fell within the copyright protection regime was not resolved at the time that computer software was reaching a state where it began to resemble a work of authorship, legislation, treaties, and to a certain extent national court decisions, have firmly established that copyright protects such works, in any form they take.[16]

Copyright gives exclusive rights to the author to have dominion over certain activities of others when those activities involve the works to which the author has received copyright protection. A recitation of verbs articulates the actions of others over which the copyright holder may either grant permission to do (via licence), or prevent from doing (via enforcement actions) (see Table 3.1).

National laws articulate these rights using slightly different terminology, although doing so generally in conformance with the Berne Convention framework; because many of the Open Source licences in common use were first drafted in the US, they often use the terminology from US copyright law (together with, or as an alternative to, the Berne Convention formulation). For this reason, any particular Open Source licence may have a degree of ambiguity as to whether all relevant copyright rights are conveyed, expressly or by implication; as shown in Table 3.1 above, there is a potential concordance of, *inter alia*, the verbs used in US copyright law with the verbs used in other national law or international treaties.

Open Source licences all express their permissions using at least some of these verbs,[17] and virtually all attach certain conditions (even if fairly minimal and easy to satisfy, as with permissive licences) to those permissions so as to keep the licensed code Open Source, or at least allow users to understand the code is based upon Open Source code, and possibly to seek out and receive the corresponding source upon which the code is based.

[15] Berne Convention for the Protection of Literary and Artistic Works (9 September 1886; as revised through 28 September 1979).

[16] It is notable that the Berne Convention does not specify that software falls within its ambit. See Geoffrey S Kercsmar, 'Computer Software & Copyright Law: The Growth of Intellectual Property Rights in Germany' (1 May 1997) 15(3) *Penn State International Law Review* article 7, at 567. Subsequent treaties, however, have established that Berne Convention compliance does include providing copyright protection to software. See Michael Lehmann, 'TRIPS, the Berne Convention, and Legal Hybrids' (December 1994) 94(8) *Columbia Law Review* 2621, at 2625.

[17] Note that many of the older licences may not use the term 'copyright' when authorising activities. Nevertheless, using verbs that come from national laws, or international treaties, governing copyright, would generally be interpreted to confer copyright rights. The extent to which leaving out some of the copyright verbs from a licence permission should be interpreted as a reservation of that right by the author, or instead the non-recited verbs should be understood by implication also to be granted, is an unresolved issue with some Open Source licences. See, e.g., Andrew Sinclair, 'Licence Profile: BSD' (2015) 1(1) *Journal of Open Law, Technology & Society* 1, at 3.

Table 3.1 Potential Concordance of Copyright Verbs in Various Treaties and National Laws

Berne Convention[a]	**UK Copyright Act**[b]	**EU Copyright Directive**[c]	**US Copyright Law**[d]
Translating	Adapting	Translating	Preparing Derivative Works
Reproducing	Copying	Reproducing	Reproducing
Performing	Performing		Performing
Broadcasting	Showing, Playing, Communicating	Distributing	Distributing, Displaying
Reciting			
Communicating	Issuing copies	Distributing	Distributing
Adapting	Adapting	Adapting, Altering	Preparing Derivative Works
Arranging		Arranging	Preparing Derivative Works

[a] See Berne Convention, see note 15, arts 8, 9, 11, 11*bis*, 11*ter*, 12, 14.

[b] See UK Copyright, Designs and Patents Act 1988, §§ 16–21.

[c] See Directive 2009/24/EC of the European Parliament and of the Council of 23 April 2009 on the legal protection of computer programs (Codified version) (2009).

[d] See 17 USC §106 (2010).

3.1.3 Exceptions to the author's rights in software copyright

3.1.3.1 Non-copyrightability

Although the copyrightability of software, in both source and binary form, has been well-established worldwide since at least 1993,[18] there has always been a tension in providing copyright protection to software given that software is composed of a substantial amount of material that is highly functional.[19] The European Union has made clear in its own software directive that:

> [O]nly the expression of a computer program is protected … ideas and principles which underlie any element of a program, including those which underlie its interfaces, are not protected by copyright under this Directive. In accordance with this principle of copyright, to the extent that logic, algorithms and programming languages comprise ideas and principles, those ideas and principles are not

[18] See Lehmann, 'TRIPS, the Berne Convention, and Legal Hybrids', note 16, at 2625.

[19] *Navitaire Inc. v Easyjet Airline Co* [2004] EWHC 1725 (Ch).

> protected under this Directive. In accordance with the legislation and case-law of the Member States and the international copyright conventions, the expression of those ideas and principles is to be protected by copyright.[20]

The US has a similar concept, the so-called idea/expression dichotomy, established via a long line of court decisions.[21]

Navigating the distinction between that in software which is not copyrightable (and thus available for non-authors to use without adhering to the conditions of any particular licence), and that which is, is not an insubstantial task. The European Court of Justice (ECJ) addressed this issue in 2012 *SAS Institute* case,[22] stating 'the ideas and principles which underlie any element of a computer program, including those which underlie its interfaces, are not protected by copyright'.[23] A similar issue in the US was the subject of a long-standing and significantly publicised legal dispute, eventually resolved by the Supreme Court of the US.[24] That dispute involved an allegation by Oracle that Google copied certain application programing interfaces (APIs), and possibly other information,[25] from Oracle's Java programs without the benefit of a licence from Oracle. Google was successful at the trial court level in arguing that everything they copied was non-copyrightable,[26] while Oracle was successful at the intermediate appeal court level in arguing that some of what was copied was copyrightable.[27] Subsequently, the case was sent to the trial court, where a jury determined that Google's copying fell within the 'fair use' exceptions to US copyright law.[28] That determination was also overturned by the intermediate appeal court.[29] Thus, the US Supreme Court was presented with two questions: (i) were the interfaces copied by Google copyrightable at all, and (ii) if they were copyrightable, was Google's copying nevertheless permissible as 'fair use' under US copyright law?[30]

[20] 'Directive 2009/24/EC of the European Parliament and of the Council of 23 April 2009 on the legal protection of computer programs' Official Journal of the European Union, L 111/17, no. 11 (5 May 2009).

[21] See *Baker v Selden*, 101 US 99 (1879); *Mazer v Stein*, 347 US 201 (1954).

[22] See *SAS Institute Inc. v World Programming Ltd*, ECLI:EU:C:2012:259 (2 May 2012).

[23] *SAS Institute*, see note 26, at 31.

[24] See *Google LLC v Oracle Am., Inc.*, 593 US _ , 141 S. Ct. 1183, Docket No. 18–956, (2021).

[25] Part of the dispute between Google and Oracle, and part of the reason why the trial court and intermediate appeal court rendered differing decisions on copyrightability, relates to whether what Google copied was necessary to interface with Java, or not. *See Oracle America, Inc. v Google LLC*, 886 F.3d 1179 (Fed. Cir. 2018) (analysis of the necessity of 11,500 lines of copied code to allow interoperability).

[26] *Oracle Am., Inc. v Google LLC*, 872 F. Supp. 2d 974 (N.D. Cal. 2012).

[27] *Oracle Am., Inc. v Google LLC*, 750 F.3d 1339 (Fed. Cir. 2014).

[28] David Goldman, 'Jury sides with Google in billion dollar Oracle suit', *CNN Business* (26 May 2016) <https://money.cnn.com/2016/05/26/technology/google-oracle/index.html> accessed 6 April 2021.

[29] *Oracle America, Inc. v Google LLC*, 886 F.3d 1179 at 106 (Fed. Cir. 2018).

[30] *Google LLC v Oracle America, Inc.*, Docket No. 18–956, Petition for a Writ of Certiorari (US 24 January 2019).

Unlike the ECJ in the *SAS Institute* decision, the US Supreme Court in *Google v Oracle* avoided squarely addressing the issue of the copyrightability of software interfaces:

> Given the rapidly changing technological, economic, and business-related circumstances, we believe we should not answer more than is necessary to resolve the parties' dispute. We shall assume, but purely for argument's sake, that the entire [Oracle] Java API falls within the definition of that which can be copyrighted. We shall ask instead whether Google's use of part of that API was 'fair use.'[31]

Analysing the question of whether Google's reproduction of certain interfaces[32] from Oracle's Java SE program was fair use, the US Supreme Court ultimately decided that Google's reproduction fell within the 'fair use' provisions of US copyright law.[33] Although the decision is complex, and goes through a detailed analysis of all the statutory fair use factors, perhaps the two most important aspects of the decision analysing whether the use of software interfaces are subject to copyright claims by the authors of those interfaces related to the questions of 'transformative use' and the manner in which future courts may decide fair use questions in software copyright disputes. On the question of 'transformative use', the Court in *Google v Oracle* stated:

> Here, Google's use of the [Oracle] Java API seeks to create new products. . . . To the extent that Google used parts of the [Oracle] Java API to create a new platform that could be readily used by programmers, its use was consistent with the creative 'progress' that is the basic constitutional objective of copyright itself.[34]

Thus, it could well be that, under the *Google v Oracle* fair use test, most reproductions of software interfaces in the US in order to create new, interoperable but noncompetitive software will be found to be not subject to the copyright claims of the authors of those interfaces, because such uses will be determined to be 'transformative'. If so, the outcome in the US on software interfaces may not be much different than the outcome in the EU under the *SAS Institute* decision. Future decisions in

[31] *Google LLC v Oracle America, Inc.*, 593 US ___, 141 S. Ct. 1183, Docket No. 18-956, Opinion of the Court at 15 (5 April 2021). The dissenting opinion in that decision criticised the majority's failure to address the copyrightability question. *Google LLC v Oracle America, Inc.*, see note 34, Dissenting Opinion at 1–2.

[32] Although the *Google v Oracle* decision often refers to 'Java APIs' and 'interfaces', the discussion focuses exclusively on Google's reproduction of the method, class, and package structures used in certain parts of Java most useful in smartphones for programers familiar with the schema in Java. *Google LLC v Oracle America, Inc.*, 593 US ___, 141 S. Ct. 1183, Docket No. 18-956, Opinion of the Court at 26 (5 April 2021).

[33] *Google LLC v Oracle America, Inc.*, 593 US ___, 141 S. Ct. 1183, Docket No. 18-956, Opinion of the Court at 1 (5 April 2021).

[34] *Google LLC v Oracle America, Inc.*, see note 37, at 25.

the US applying the fair use analysis to different software interface scenarios may clarify if this is indeed the case.[35]

The *Google v Oracle* decision also clarified an important point on 'fair use' analyses in copyright disputes:

> 'In this case, the ultimate "fair use" question primarily involves legal work. "Fair use" was originally a concept fashioned by judges … Our cases still provide legal interpretation of the fair use provision. And those interpretations provide general guidance for future cases.'[36]

This finding is an important procedural point in US law as it may result in early hearings in software copyright disputes to resolve the 'primarily legal' question of fair use. Early hearings on the legal question of patent claim interpretation have now become a regular part of patent litigation practice in the US.[37] Such procedures, if adopted in the future, could have curtailed the nearly eleven-year litigation history of the *Google v Oracle* dispute. Nevertheless, unless there are future developments in the US on the question of software copyrights in general, or software copyright interfaces in particular, there will be greater ambiguity about the use of such interfaces, and the effort required to resolve them, in the US than there currently is in the EU.

The question of what parts of software are subject to copyright protections and which are free for anyone to use without fear of a claim of copyright infringement or licence violation is an important issue when analysing the effect of Open Source licences on downstream recipients. This is particularly the case with copyleft licences, as the downstream recipient may not need to follow the licence's requirement to use the same licence, if they are using only material for which either copyright protection does not extend, or for which the right of 'fair use' or 'fair dealing' (or equivalents) applies. In the EU, the *SAS Institute* case provides some concrete guidance on this question; the courts in the US have not resolved this issue as conclusively, as the US Supreme Court's decision in *Google v Oracle* left the question to a case-by-case, legal analysis somewhat dependent upon underlying facts. As a result, distributors

[35] An early, and thoughtful, analysis of this question—written after the *SAS Institute* decision but almost a decade before the ultimate decision in *Google v Oracle*—may be found at Walter van Holst, 'Less May Be More: Copyleft, -Right and the Case Law on APIs on Both Sides of the Atlantic' (2005) 5(1) *Journal of Open Law, Technology & Society* at 5 <https://jolts.world/index.php/jolts/article/view/72/143> accessed 6 April 2021. In that analysis, the author concludes that ' [w]hen taking the most recent jurisprudence on software APIs [i.e., *SAS Institute*] into account, one can argue that the LGPL is not really the Lesser GPL, but that the GPL is based on a by now outdated understanding of software copyright and effectively becomes equal to the LGPL.' Von Holst, 'Less May Be More', at 13. This conclusion may also follow if the *Google v Oracle* decision is ultimately determined to be very near to the outcome in *SAS Institute*.

[36] *Google LLC v Oracle America, Inc.*, 593 US ___, 141 S. Ct. 1183, Docket No. 18-956, Opinion of the Court at 19 (5 April 2021).

[37] Edward Brunet, 'Markman Hearings, Summary Judgment, and Judicial Discretion' (2005) 9(1) *Lewis & Clark Law Review* 93, at 95–96.

and recipients of code under Open Source licences may be forced to analyse the question of whether the use of software interfaces require adherence to the Open Source licence terms depending on whether the use is within, or without, the US. If the use is in the US, a detailed factual analysis of the circumstances in which software interfaces are reproduced, under the fair use tests under US copyright law,[38] may be required—at least until such time as future court decisions provide greater guidance on when reproduction of software interfaces is not 'fair use'.[39]

3.1.3.2 Functional dictation and merger doctrine

Authored material that would otherwise be copyrightable expression has nevertheless been found not to be subject to copyright protection when the expression is directed to an idea that is incapable of being expressed, as a practical matter, in more than one or a small number of ways. In the UK, this concept has been established under the rationale 'that if expression is dictated by technical function then the criterion of originality [required for copyright protection] is not satisfied'.[40] In the US, this concept is designated the 'merger doctrine'—that the copyrightable expression in a work of authorship has 'merged' with the non-copyrightable idea being expressed, when that idea is incapable of being expressed, as a practical matter, in more than one or a small number of ways.[41] Although in many ways non-copyrightability and functional dictation/merger doctrine cover similar territory and rely on related statutory bases and case law, the latter doctrine does admit of the possibility of arguing that—even though there may be multiple ways of expressing certain ideas, facts, systems, or processes through code—to the extent that those multiple ways are unduly constraining on the ability of other authors to capture those ideas, facts, systems, or processes without running afoul of a different author's copyrights, protection under copyright should not be afforded.

[38] 17 USC § 107.

[39] Although the newness of the *Google v Oracle* decision has meant there is limited commentary on its potential import on software development practices in the US, at least one commenter has indicated that reproduction of code for interoperability may be found to be fair use given recent court decisions, including *Google v Oracle*, emphasising transformative uses of copyrighted materials as being more often than not fair. 'Google's Supreme Court win could actually benefit the little guy' MarketPlace (6 April 2021) <https://www.marketplace.org/shows/marketplace-tech/googles-supreme-court-win-could-actually-benefit-the-little-guy-oracle-java/> accessed 6 April 2021.

[40] See *SAS Inst. Inc. v World Programming Ltd.* [2013] EWCA Civ 1482, [31]–[33], available at <http://www.bailii.org/ew/cases/EWCA/Civ/2013/1482.html> accessed 13 April 2022.

[41] See Pamela Samuelson, 'Reconceptualizing Copyright's Merger Doctrine' (2017) 63 *Journal of the Copyright Office Society of the United States of America* 417–70, available at <https://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID2763903_code1160955.pdf?abstractid=2763903&mirid=1&type=2>. Professor Samuelson argues that merger doctrine in the US is not just limited to preventing 'ideas' from being captured and restricted as copyrightable expression, but that instead there are other dichotomies that the merger doctrine also covers, including 'fact/expression', 'process/expression', 'system/expression', Samuelson, 'Reconceptualizing Copyright's Merger Doctrine' at 417–70. Each of these dichotomies represent a possible limitation on the ability to claim copyright protection on certain aspects of computer software.

3.1.3.3 'Fair dealing' and 'fair use'

The law in both the UK and the US allows for certain 'fair' exercises of the exclusive rights of a copyright holder without subjecting those exercises to infringement liability. In the UK, this exception to the holder's rights is called 'fair dealing';[42] in the US, the exception is called 'fair use'.[43] Despite the similarities in the names, the effect—particularly in the area of software—can be quite different. In the US, 'fair use' has been the foundation of a number of defences to claims of copyright infringement for software,[44] and in fact fair use was the basis upon which the *Oracle v Google* dispute was decided by the Supreme Court of the US.[45] Fair use as a defence to a claim of copyright infringement in the US is defined in statute and is analysed using a multifactored factual analysis not specific to the use that is being made; a court must analyse:

- the purpose and character of the use, including whether such use is of a commercial nature or is for non-profit educational purposes;
- the nature of the copyrighted work;
- the amount and substantiality of the portion used in relation to the copyrighted work as a whole; and
- the effect of the use upon the potential market for or value of the copyrighted work.[46]

In contrast, fair dealing in the UK is much more specific to the type of use being made, and specifically excludes many uses of computer software from its ambit.[47] Instead of relying upon fair dealing to address certain 'fair' uses of computer software, the UK Copyright Act sets forth, in sections separate from the defined uses under fair dealing, specific acts relative to computer software which are explicitly excluded from infringement:[48]

- making any back up copy;
- converting it into a higher-level language;
- decompiling it to obtain the information necessary to create an independent program which can be operated with the program so decompiled, or with another program;

[42] UK Copyright Act, see note 18, §§ 29.

[43] 17 USC § 107 (1992).

[44] See, e.g., *Sega Enterprises Ltd v Accolade Inc.,* 977 F.2d 1510 (9th Cir. 1992).

[45] *Google LLC v Oracle Am., Inc.*, 593 US ___, Docket No. 18-956, Opinion of the Court at 1 (5 April 2021).

[46] 17 USC § 107 (1992).

[47] UK Copyright Act, see note 18, § 29 (excluding converting a computer program to a higher-level language, copying incidental to such a conversion, and observing, studying or testing the functioning of a computer program in order to determine the ideas and principles which underlie any element of the program).

[48] UK Copyright Act, see note 18, §§ 50A–50C.

- observing, studying, or testing the functioning of the program in order to determine the ideas and principles which underlie any element of the program;
- copying it or adapting it for the purpose of correcting errors in it.

In summary, US law admits of perhaps a broader range of acts concerning software that might be considered 'fair' and outside of the threat of infringement claims, but in order to establish that such use is 'fair', the accused infringer is subject to a detailed factual analysis amenable to differing interpretations. In the UK, in contrast, the activities considered 'fair' are much more narrowly circumscribed, but are potentially much easier to establish factually.

3.1.4 Derivative works in software copyright

3.1.4.1 A brief summary of computer software architecture and interactions

The many ways in which a particular piece of software can be configured to interact with other pieces of software is far beyond the scope of this chapter, and is sufficiently malleable with new developments in technology, that it would be virtually impossible to discuss all the different ways such interactions can occur, and the licensing and copyright implications that result. Nevertheless, the process of writing source code, converting that source code into executable code, delivering that executable code to a target computing device, and executing that executable code on that target device, often fall into certain general techniques that it is of use to describe them in brief to discuss common use cases that frequently raise questions with regard to the effect of certain Open Source licences. Figure 3.1 represents a common software build scenario.

One or more modules A and B are written in source code by human programers. A software tool called a compiler is used to both convert the source modules into a format called object code, and (possibly) to intermingle parts of A and B together into a unitary object module AB. There may also be standard libraries or other pre-existing code modules which the object module AB is designed to make use of.

These libraries can be used by the compiled code AB in one of two ways—via a static link or a dynamic link. In a static link, the object code AB is input into a tool called a linker, together with object code of the library SL, to form a unitary binary ABSL, all of which is distributed to a user to be executed or run on their computing machine. In a dynamic link, the object code AB is distributed to the user under the assumption or expectation that the library which the object code AB is designed to use is already installed on the user's computing machine or will be separately obtained by the user. The combination of AB and the library DL is not made until 'run-time', in other words the two are combined together in the computing machine's memory upon execution of the object file AB.

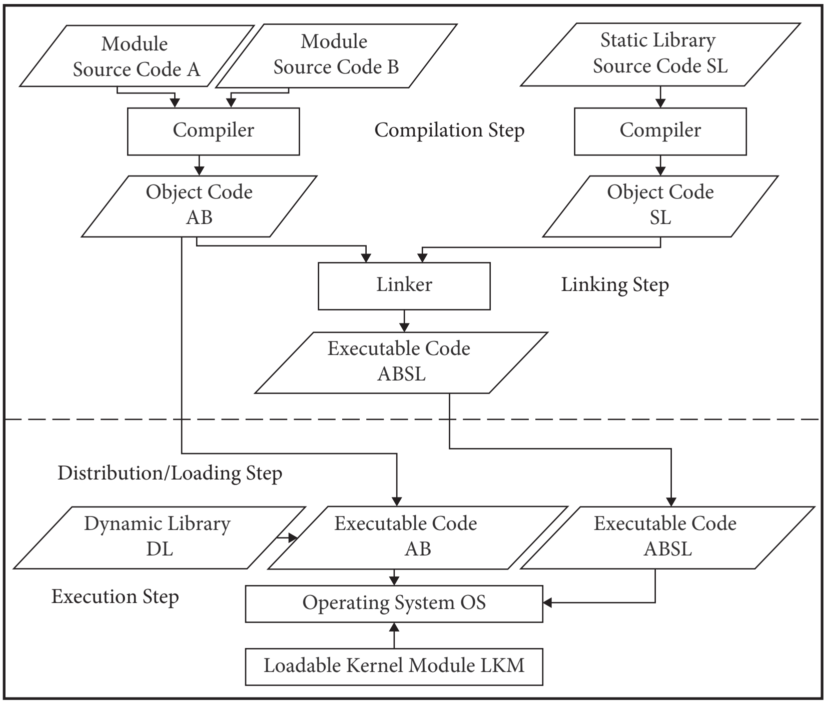


Figure 3.1 Common Software Build Scenarios

There is an additional piece of software that may be used for which free and open source licensing issues are implicated. A loadable kernel module (often abbreviated as LKM) is a piece of code (in most cases, drivers written to allow the kernel of the operating system—which manages the computing machine's resources—to make use of certain components within the computing machine) that is dynamically loaded into the operating system kernel at execution, rather than integrated into the kernel itself.

The concepts outlined above may be of importance to understand when interpreting the impact of certain free and open source licensing, particularly the copyleft licences which purport to apply their terms to programs which may be combined or linked, as discussed in more detail in section 3.2.2 later in this chapter.

3.1.4.2 Derivative works and Open Source

A number of Open Source licences use the term 'derivate works' to define the scope of certain activities related to the modification, adaptation, or translation of the licensed code.[49] As discussed earlier in section 3.1.2, 'derivative works' is

[49] See, *inter alia*, GPLv2, § 0.

a concept from US copyright law which encompasses things like modification, translation, adaptation, and arranging in international copyright treaties or non-US national laws, but it does not have universal meaning that is consistent across national laws,[50] or even within courts in the US.[51] The boundaries of what is or is not a derivative work, or is equivalent or corelated in non-US law, is also circumscribed by the exceptions to copyright protection and enforceability set forth in section 3.1.3 earlier in this chapter. Because of this, any interpretation of the rights granted under an Open Source licence, or the obligations attached to that grant, will be subject to potential ambiguity to the extent it references derivative works without further clarifying what types of derivative works are governed by the terms of the licence. More detail on how that ambiguity works in practice can be found in section 3.2.2 later in this chapter.

3.2 Forms of Open Source Licensing

As discussed in more detail in Chapter 1, Open Source licensing is—at its heart—a philosophical movement which seeks to upend or reverse the more conventional 'proprietary' model where the rights required to make productive use of some useful thing are granted restrictively, and in most cases without any rights to study, learn, and adapt the underlying architecture or design of that thing. In order to put into practice this philosophy, there must be legal rights granted, and those rights need to be granted in a way that is consistent with that philosophy. One of the most important forms of legal rights used to achieve this aim is copyright—specifically, the copyright in both the source and executable forms of software discussed in more detail in section 3.1.2 earlier in this chapter.

In the late 1980s,[52] two alternative ways of achieving the overall aims of the free and open source philosophy were (roughly) simultaneously created: permissive licensing and copyleft licensing. Each of these models have numerous variants, and at least copyleft has two generally recognised sub-variants—'strong' copyleft and 'weak' copyleft, but both models share certain common characteristics and in certain cases can be used in a complementary way. Practitioners are nevertheless cautioned that there are also significant incompatibility issues between some of these

[50] Till Jaeger, 'Enforcement of the GNU GPL in Germany and Europe' (2010) 1 *Journal of Intellectual Property, Information Technology and E-Commerce Law* 34, at 36.

[51] Omar Johnny, Marc Miller, and Mark Webbink, 'Copyright in Open Source Software—Understanding the Boundaries' (2010) 2(1) *Journal of Open Law, Technology & Society* 13, at 24.

[52] The earliest examples of licences satisfying both the Free Software Definition and the Open Source Definition, the MIT licence, the BSD licence, and the GPL licence, date—at their earliest instantiation—from 1987, 1988, and 1989, respectively. See Gordon Haff, 'The Mysterious History of the MIT License' opensource.org (26 April 2019), <https://opensource.com/article/19/4/history-mit-license> accessed 19 January 2020; Richard Stallman, 'New General Public License' (February 1989) <https://groups.google.com/forum/#!msg/gnu.announce/m0Jjj_64PeQ/8xL1xkVKJb8J> accessed 9 March 2020.

licences, and understanding when and where particular licences can be used in a complementary way, and where such licences create unresolvable conflicts, is very much dependent upon the text of the particular licences used, the state of copyright law in the particular jurisdiction where the licence might be enforced, a good understanding of the particular programing paradigm being used, and how that programing paradigm maps to the licence texts at issue and copyright law in the particular jurisdiction in question.

Most free and open source licences allow the user to make ‘private’ use of the software—meaning that anything the user does without allowing access to others imposes no legal obligations on that user.[53] It is when the user seeks to allow others access to that software—typically by distributing the software, or a modified version of the software—that the licence imposes legal obligations on that user. Many of those obligations are similar amongst all licences, whilst others differ in standard ways that allow categorisation and subcategorisation between licences. Those categorisations and sub-categorisations are discussed in more depth in the following sections.

3.2.1 Permissive licensing

3.2.1.1 The BSD and MIT licences

The BSD[54] and MIT[55] licences are the oldest free and open source licences still in use to this day. Both are quite similar in the way that they are structured and the obligations that they impose upon the recipient, although there are some subtle differences that might cause an author to choose one over the other, or for a court or arbiter to determine that they have different legal effect. Both licences grant broad licences, at least under copyright. Both oblige preservation of copyright notices for those who exercise the licence grants. Both oblige that a copy of the licence text be provided for any exercise of the licence grants. Both disclaim liability on behalf of the authors.

[53] The Free Software Foundation’s so-called Freedom Zero— ‘The freedom to run the program as you wish, for any purpose’—generally contemplates this concept. See Free Software Foundation, ‘What is Free Software? The Free Software Definition’ <https://www.gnu.org/philosophy/free-sw.html.en> accessed 29 February 2020. As noted later in the chapter, certain Open Source licences test this concept.

[54] There is no single ‘BSD’ licence; there are numerous variants, generally differentiated by the number of clauses they contain. Thus, 4-Clause BSD <https://spdx.org/licenses/BSD-4-Clause.html>, 3-Clause BSD <https://spdx.org/licenses/BSD-3-Clause.html>, 2-Clause BSD <https://spdx.org/licenses/BSD-2-Clause.html>, 1-Clause BSD <https://spdx.org/licenses/BSD-1-Clause.html>, and 0-Clause BSD <https://spdx.org/licenses/0BSD.html> accessed 21 July 2022. The 4-Clause BSD contains an ‘advertising clause’ which is generally considered to create compliance issues, and is thus generally disfavoured. See Sinclair, ‘Licence Profile: BSD’, see note 21, at 4–5.

[55] There is also some dispute as to whether there is more than one ‘MIT’ licence. See GNU Operating System, ‘Various licenses and comments about them’, <https://www.gnu.org/licenses/license-list.html#Expat> accessed 10 March 2020. The version approved by the Open Source Initiative is generally considered the canonical version, and is the version discussed later in the chapter. See <https://opensource.org/licenses/MIT>.

The effect of both licences is to allow recipients to exercise the author's copyright rights, without imposing any obligations on the recipients to use the same licence for any further downstream distribution.[56] They are thus *permissive* in the way in which they allow alternative licensing models to be exercised by recipients—including, restrictive, 'proprietary' licensing, and copyleft licensing.

One distinction between the two licences is that the MIT license expresses its grant using a non-statutory term ('deal in'), and then recites US statutory verbs as non-limiting examples: '[p]ermission is … granted … to *deal in* the Software without restriction, *including without limitation* the rights to use, copy, modify, merge, publish, distribute, … and/or sell copies' (emphasis added), whereas the BSD license uses fewer verbs, and uses them directly in the grant: '*[r]edistribution* and *use* …, *with or without modification*, are permitted' (emphasis added). Another distinction is that the MIT license explicitly allows sublicensing, whereas the BSD license, if it does so, does so indirectly—using the term 'redistribution'. The extent to which these distinctions are meaningful has yet to be adjudicated, and the two licences are generally considered to be relatively interchangeable in the permissions they give and the obligations they impose on those permissions.

3.2.1.2 The Apache licence

In 2000, the ASF published an alternative form of a permissive licence; that licence is now on its second—and by far most commonly used—iteration, the Apache Software License 2.0 (the Apache 2.0). The Apache 2.0 license, drafted more than a decade after the BSD and MIT licenses came into being, was designed to address some perceived ambiguities or missing features in those two licences, in particular a more robust set of definitions of the licence grants themselves[57] (and the parties to whom they extend) express granting of patent rights (with the inclusion of a patent 'defensive suspension' clause to revoke patent rights to those entities taking assertive patent actions against the licensed software) and other features designed to make the licence terms more consistent with generally

[56] There is a fairly small minority of users of these licences who have argued that the requirement to *provide* a copy of the licence obliges the recipient to *use* that licence, thus rendering these licences copyleft. See Anonymous Coward, 'Theo de Raadt on Relicensing BSD Code', *OpenBSD Journal* (13 September 2007) <http://undeadly.org/cgi?action=article&sid=20070913014315> accessed 13 April 2022. The general consensus is, however, to the contrary and re-licensing under different terms—including restrictive proprietary licences and copyleft licences—is a common practice with permissively licensed software.

[57] Of particular interest is the manner in which the Apache 2.0 licence defines the scope of modifications/derivative works that are subject to the Apache 2.0 licence terms. The Definition section of the Apache 2.0 licence states that '[f]or the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.' Given the overall permissive nature of the licence, this clarification of the scope of the grant may be in practice relatively insubstantial, although it could be an important distinction in some architectural scenarios given that avoiding the application of the Apache 2.0 licence to certain works may be useful because of the incompatibility of the Apache 2.0 licence with certain of the GNU family of licences. See section 3.3.2.3 later in this chapter.

accepted licence drafting standards. Nevertheless, the overall effect of the copyright licence grants in the Apache 2.0 licence are intended to be the same as those of the BSD and MIT licenses discussed earlier, requiring preservation of copyright notices and providing a copy of the licence with the code, but otherwise permissively allowing the recipient of the code to use terms of its own choosing, including proprietary licensing, or copyleft licensing, when that recipient exercises the granted copyright licences. As discussed in more detail in section 3.3.2.3, there are certain caveats to the overall permissive nature of the Apache 2.0 licence: the GNU family of licences. The Free Software Foundation (FSF) has stated that the Apache 2.0 license is incompatible with GPLv2;[58] although the ASF disagrees with this assessment, it nevertheless states that with regard to use of GPLv2, 'you should always try to obey the constraints expressed by the copyright holder when redistributing their work'.[59] This incompatibility introduces certain complications in architecting software stacks that may include Apache 2.0 and GPLv2 code (in particular, the Linux kernel of the GNU/Linux operating system, which is licensed under GPLv2), and could very well be the reason why the more recent, and more carefully drafted, Apache 2.0 license has not supplanted the continued popularity of the BSD and MIT licenses as *de facto* choice when selecting a permissive licence.

3.2.1.3 Other permissive licences

There are a large number of permissive licence variants currently listed as either approved by the Open Source Initiative (OSI) or determined to be free software licences by the FSF. The vast majority of these licences are variants of the BSD (which itself has numerous variants) or MIT licenses, and a handful of Apache 2.0 variants as well, and thus would generally operate in a manner similar to those licences. Nevertheless, when confronted by these numerous variants, it is advisable to read their terms carefully as some variants may introduce additional complications or incompatibilities.[60] Two other permissive licences that have some degree of popularity and which practitioners may encounter and thus need to familiarise themselves with are: the Academic Free License[61] and the Artistic License.[62] The

[58] See FSF Licence Comments, see note 59, <https://www.gnu.org/licenses/license-list.html#apache2> accessed 21 July 2022. Note that by the logic used in the FSF's commentary on the Apache 2.0 licence, Apache 2.0 is likely also to be incompatible with the Affero GPLv1.0 and 2.0 licences, and partially incompatible with the Lesser GPLv2.1 licence.

[59] Apache Software Foundation, 'GPL Compatibility' <https://www.apache.org/licenslicences/GPL-compatibility.html> accessed 10 March 2020.

[60] For example, licences with so-called advertising clauses may be incompatible with many other Open Source licences. See GNU Operating System, 'The BSD License Problem' <https://www.gnu.org/licenses/bsd.html> accessed 10 March 2020.

[61] Open Source Initiative, 'Academic Free Licence ("AFL") v. 3.0' <https://opensource.org/licenses/AFL-3.0> accessed 10 March 2020.

[62] Open Source Initiative, 'Artistic Licence v. 2.0' <https://opensource.org/licenses/Artistic-2.0> accessed 10 March 2020.

Academic Free License is designed to be somewhat similar in operation to the Apache 2.0 license in that it has more robust drafting and has specific reference to patent rights; it is also notable in that its author has explicitly stated that that licence is designed to operate as a contract,[63] thus explicitly settling the 'bare licence versus contract' debate—discussed in section 3.4 later in this chapter—for that licence. The Artistic Licence is notable in that it was the subject of one of the first, and still one of the leading, cases in the US examining the operation of Open Source licensing.[64] That decision, at a minimum, demonstrates that although permissive licences are designed to be easy to comply with and to allow flexibility to the recipient in modifying and combining code while also being able to choose its own licence, failure to comply with even the simplest licence conditions in a permissive licence can result in enforcement action and could be found to be a violation of the author's copyright rights.[65]

3.2.2 Copyleft licensing

The second broad category of Open Source software license types are the copyleft[66] licences. In all but one very important way, copyleft licences are designed to operate in a way similar to the permissive licences—a broad grant of copyright rights (and in virtually every copyleft licence, at least an attempt to do the same with patent rights), a requirement to preserve copyright notices, and a requirement to provide a copy of the licence with the code. The important distinction is that copyleft licences require certain exercises of the author's copyright rights to be licensed under identical terms. Thus, copyleft licences put limitations on the recipient's ability to use other licensing models (e.g. proprietary, or permissive, or even a different copyleft licence) for their own downstream activities. Copyleft licences are often broken into two subcategories: 'strong' copyleft and 'weak' copyleft. 'Strong' copyleft licences, in general, are designed to have fewer exclusions[67] to the requirement to use the same licence when exercising the author's copyright rights, whereas

[63] Lawrence Rosen, 'Open Source Licensing: Software Freedom and Intellectual Property Law' (Prentice Hall: Upper Saddle River, NJ, 2005) at 181.

[64] *Jacobsen v Katzer*, 535 F.3d 1373 (Fed. Cir. 2008).

[65] *Jacobsen v Katzer*, note 64, at 1382.

[66] 'Copyleft' is a coined term intended to be a pun on 'copyright', contrasting the highly permissive terms of that family of licences with the generally perceived highly restrictive nature of copyright protection and copyright licensing in the software industry. An additional pun for copyleft, 'all rights reversed', is a play on the now-obsolete notification that used to often accompany copyrighted works, 'all rights reserved'.

[67] Although strong copyleft licences are generally designed to require the use of the same licence when the recipient exercises the author's copyright rights, even the strongest of copyleft licences (GPLv2, GPLv3, AGPLv1, and AGPLv3) all allow the recipient to engage in some form of private modification of the code without triggering the obligations of those licences. See, e.g., GPLv3 § 0 (definition of 'propagate').

'weak' copyleft licences are designed specifically to articulate circumstances where an exercise of the author's copyright rights is permitted while allowing for alternative licensing (again, be it permissive, or proprietary, or a different copyleft licence) of code resulting from that exercise. The strong copyleft licences typically require that derivative works (or some other formulation intended to capture that concept under both the law of the US and other jurisdictions) cannot be distributed under any other licence.

3.2.2.1 The GNU family of licences

Copyleft licensing was pioneered by—and the term 'copyleft' was indeed coined by—the FSF. The FSF maintains a family of copyleft licences—all under the 'GNU'[68] moniker—all intended to allow different forms of copyleft to be used, depending on the degree of copyleft obligation to be imposed on the user. This family of licences can generally be divided into the 'strong copyleft'—GPLv2, GPLv3, AGPLv1 and AGPLv3—and 'weak copyleft'—LGPLv2.1 and LGPLv3.0—variants.

3.2.2.1.1 The GNU General Public Licence (GPL)

3.2.2.1.1.1 GPLv2 The GNU General Public License, version 2 ('GPLv2') is a strong copyleft licence first published by the FSF in 1991. It grants the right to exercise enumerated copyright rights (copy, distribute, and modify the software), under the condition that the resulting software is again distributed[69] under the identical conditions of GPLv2. As with other Open Source licences, GPLv2 requires including the GPLv2 licence text, providing the source code for any distributed executables, and making reference to the disclaimer of warranty. GPLv2 states that failure to follow the licence terms results in the revocation of the licence, although third parties (such as downstream recipients) are unaffected by such failure. The obligation to grant access to the source code in case of distribution of executable copies requires providing 'Complete Corresponding Source Code', defined as 'all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable'.[70]

GPLv2, by its own terms, extends the obligations to a 'work based on the Program', defined as 'any derivative work under copyright law; that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language'.[71] However, GPLv2 also states that 'mere

[68] 'GNU' is a recursive acronym for 'GNU's not UNIX', an express acknowledgement that the FSF was attempting to create—using the GNU family of licences—a computer operating system intended to be an alternative to the then-ubiquitous UNIX operating system.

[69] It is notable that the licence obligations in GPLv2 are attached upon 'distribution' of code covered by that licence; the licence itself makes clear that merely running the program can be done without need to follow the licence obligations. See GPLv2, § 0.

[70] GPLv2, § 3. This detailed definition is intended to prevent source distributions which are not amenable to modification and compilation because of scripts or tools not otherwise available to distributees.

[71] GPLv2, § 0.

aggregation of another work not based on the Program with the Program … on a volume of a storage or distribution medium does not bring the other work under the scope of this License'.[72] GPLv2 further states that when sections of the new work

> are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.[73]

GPLv2 states that the licence is designed 'to exercise the right to control the distribution of derivative *or collective works* based on the Program';[74] thus any analysis of whether GPLv2 would apply to a particular use and distribution of code under that licence would need to consider the question of whether a derivative work or collective work has been created.[75]

As may be clear from the earlier discussion, determining what particular exercises of copyright rights under GPLv2 obligate the licensor to follow the terms of that licence is not explicitly clear; this issue is made even more thorny by the fact that which does, and does not, fall outside of the realm of 'derivative works' under US Copyright law (from whence the term 'derivative works' in the licence comes) for software is not well-defined and in fact subject to a number of different tests.[76]

GPLv2's obligations purport to apply to any derivative or collective works; anything that is outside the definition of a derivative or collective work should not be affected by the licence's obligations. According to the FSF—stewards of GPLv2—a work that links to code licensed under GPLv2 (statically or dynamically) forms part of the modified work and it must be treated accordingly.[77] This assertion is a matter of some controversy, and many commentators believe that although static linking arguably does require the linked program to be licensed under GPLv2, dynamic linking should not.[78]

[72] GPLv2, § 2.

[73] GPLv2, § 2.

[74] GPLv2, § 2.

[75] 'Collective works' are a concept under US Copyright Law, 17 USC § 101, defined as 'a work … in which a number of contributions, constituting separate and independent works in themselves, are assembled into a collective whole'. GPLv2 does not make clear the distinction between covered 'collective works' and uncovered 'mere aggregations'.

[76] See section 3.1.4.2 of this chapter.

[77] See Free Software Foundation, 'Frequently Asked Questions About the GNU Licenses' <https://www.gnu.org/licenses/gpl-faq.en.html#GPLStaticVsDynamic> accessed 29 February 2020.

[78] See, e.g., Malcolm Bain, 'Software Interactions and the GNU General Public License' (2010) 2(2) *Journal of Open Law, Technology & Society* 165, at 177.

In the end, the GPLv2 licence in part depends on the degree of current ambiguity around the scope of derivative works (or, in other jurisdictions, the acts of adapting/arranging/translating of copyrighted works). As more decisions are rendered interpreting the scope of copyright in software—such as, for example, whether certain interfaces in software are copyrightable at all[79]—or the specific provisions of 'work based on the Program' under GPLv2, more clarity may be provided as to how broadly that licence applies to other programs that may work with or around GPLv2 licensed code.

3.2.2.1.1.2 GPLv3 In 2007, the FSF published the GNU General Public License, version 3.0 (GPLv3).[80] GPLv3 was intended to modernise GPLv2 in several ways considered to be important to the FSF, including: (i) 'internationalising' the language to make it less US law-centric; (ii) adding clearer and more specific conditions and obligations around patent rights; and (iii) addressing certain behaviours considered to be antithetical to software freedom, but which it was believed were not addressed adequately in GPLv2.[81]

Many of the same rights and obligations that existed in GPLv2 were preserved in GPLv3, albeit with slightly updated language; thus, much of the discussion of GPLv2 in section 3.2.2.1.1 earlier in this chapter would apply equally to GPLv3.[82] GPLv3 does explicitly state that it intends to capture, under the obligation to provide source code, 'dynamically linked subprograms that the work is specifically designed to require',[83] thus making explicit in the licence text that which was only referenced in a FSF FAQ for GPLv2—that dynamic linking was considered by the FSF to require the program so linked to be licensed under GPLv3. As discussed earlier in section 3.2.2.1.1 with reference to GPLv2, whether dynamic linking would—under copyright law—require the linked program to be licensed according to the terms of the program to which it is linked is a matter of some dispute that is currently unresolved.

GPLv3 also attempts to address concerns raised by the passage of the Digital Millennium Copyright Act (DMCA)[84] in the US in 1998—an Act not in effect

[79] See Section 3.1.3.1 above.

[80] GNU Operating System, 'GNU General Public License' <https://www.gnu.org/licenses/gpl-3.0.html> accessed 29 June 2007.

[81] See Free Software Foundation, 'GPLv3 First Discussion Draft Rationale' (16 January 2006), <http://gplv3.fsf.org/gpl-rationale-2006-01-16.pdf> accessed 13 April 2022.

[82] One change between GPLv2 and GPLv3 that may be merely a wording clarification to capture international norms is that the licence obligations in GPLv3 are triggered upon 'conveyance', a defined term which includes an embedded defined term, 'propagation', which term includes, but is not limited to, distribution. See GPLv3, § 0. The definition of 'propagation' in GPLv3 references both direct and secondary liability under copyright law, and thus attempts to capture acts that would only be infringing on licensor's rights upon acts by third parties. The legal effect of this definitional change is as-yet undetermined, although it likely is an attempt—together with the definition of 'Corresponding Source' in § 1—to cover dynamic linking.

[83] GPLv3, § 1.

[84] 17 USC § 1201 (1999).

at the time of publication of GPLv2 in 1991. GPLv3 provides that no work covered by the licence may be deemed part of an effective technological protection measure under any applicable law;[85] since 'technological protection measures' is a term from the DMCA, this provision is aimed squarely at that law and any related laws outside the US, attempting to allow copyright holders a right of action against those circumventing various technical means designed to prevent access to the copyrighted works, such as Digital Rights Management (DRM). Given all the other obligations of GPLv3 to provide Complete Corresponding Source, as well as to license under the terms of GPLv3, it is difficult to foresee scenarios where a technological protection measure could be built into GPLv3 code to adequately invoke the DMCA or related laws. Nevertheless, the drafters of GPLv3 were sufficiently concerned that users might attempt to do so that they made explicit that it was prohibited by that licence.

GPLv3 also includes a requirement intended to prevent certain hardware manufacturers from using GPLv3 code on their devices but including technical mechanisms in that hardware—such as installation keys—to prevent the hardware user from modifying, installing, and running the GPLv3 code on that device. The 'Installation Information'[86] section of GPLv3 is complex and admits of several exceptions whereby a hardware maker would not be required to comply. Perhaps the most important exception is that it only applies to a certain subsegment of hardware devices, so-called User Products. The definition of a 'User Product' for which 'Installation Information' must be provided under GPLv3 is derived from a definition in consumer protection laws in the US;[87] the interpretive law surrounding that consumer protection law—and possibly corresponding consumer protection laws outside the US—is likely to guide a court in understanding the sorts of devices to which that provision applies. For such products, encryption keys, hardware checksums, or other technical information needed to install and operate modified GPLv3 software on such products would need to be provided as 'Complete Corresponding Source'.[88]

3.2.2.1.2 The GNU Lesser General Public Licence (LGPL)

The FSF realised that in certain circumstances, it would not be pragmatic to license all code under a 'strong' copyleft licence like one of the GPL licences. For example, certain libraries might be of greater use, and enjoy greater adoption, if they did not impose copyleft requirements on any program making use of that library. As a result,

[85] GPLv3, § 3.

[86] GPLv3, § 6.

[87] 15 US Code § 2301(1) (1975).

[88] For a detailed explanation of how the 'Installation Information' requirement of GPLv3 works and how it relates—if at all—to requirements in GPLv2, see P McCoy Smith, '"Installation Information", GPLv2 and GPLv3: What is it and what must you provide?' (27 September 2021) Linux Foundation Open Source Summit <https://www.youtube.com/watch?v=6W3LBlkOpDM&t=2s> accessed 8 June 2022.

the FSF created a licence—initially named the 'Library General Public License' but later changed to be called the 'LGPL'—designed to be a 'weak' copyleft version of GPL. Like GPL, there exist two common versions in current use—LGPLv2.1, and LGPLv3. The text of LGPLv2.1 corresponds, with notable exceptions, to the text of GPLv2, whereas the text of LGPLv3 corresponds, again with notable exceptions, to GPLv3. As such, much of the discussion above with regard to GPLv2 and GPLv3 would apply to their counterpart LGPL versions.

The first exception—found in both LGPLv2.1 and LGPLv3—states that an application may make use of the following header file elements of code licensed under LGPL without the requirements of LGPL applying to that application: 'numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length)'.[89] This first exception recognises that there may be a need for some limited use to be made of parts of those header files in order for an application to interoperate (or to be statically linked) with the LGPL code, but that such uses should not force the application itself to be subject to the obligations of LGPL. This exception thus provides a form of a defined *de minimus* or fair use/fair dealing exception to the copyleft obligations of LGPL.

The second exception—also found in both LGPLv2.1 and LGPLv3—allows the creation of 'Combined Works' (a work 'produced by combining or linking' an application with the LGPL code, which Combined Work may be licensed under 'terms of your choice', i.e., any licence terms, not just LGPL).[90] The only limitation on this exception is that the 'terms of your choice' do not 'effectively … restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications', together with requiring the source code for the LGPL portion of the Combined Work, and a notification that LGPL applies to that portion. This second exception recognises that many libraries, or other commonly-referenced software components, will need to be combined together—such as by linking—and to oblige the entire combined work to be licensed under LGPL would decrease the use of such libraries. Thus, the second exception is designed to allow any licence choice (copyleft, permissive, or even proprietary), as long as the recipient of the combined work has the ability to modify and recombine the LGPL code, and to do effective debugging of the modified and recombined code. This particular exception has never been tested in court, and the FSF's own frequently asked questions (FAQs) do not explain in detail how one would need to present a non-LGPL licensed combined work to a licensee, but at a minimum, would require providing the source code for the LGPL part of that combined work under LGPL, and at least ensuring that any facilities in the combined work—such as symbols used by symbolic debugging tools—are not removed from the object code of the application to which the LGPL part is linked.

[89] LGPLv2.1, § 5; LGPLv3, § 3.
[90] LGPLv2.1, § 6; LGPLv3, § 4.

3.2.2.1.3 The GNU Affero General Public Licence (AGPL)

The GNU Affero General Public Licence (AGPL) is the third licence in the GPL family of licences maintained by the FSF. Its intent is to close the 'ASP (application service provider) loophole' in GPL: namely, that because GPL's obligations only apply to code that is either distributed (GPLv2) or conveyed (GPLv3), entities offering Software as a Service ('SaaS') under either GPLv2 or GPLv3, such that the software is accessed by third parties—but only over a network such that the code is never distributed to them—are not obliged to supply source code to those third parties. AGPL closes this 'loophole' by adding an additional condition triggering an obligation to provide source:

> if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software.[91]

The AGPL exists in two versions: AGPLv1,[92] which contains terms similar to GPLv2 (but for the additional condition recited earlier) and AGPLv3, which contains terms similar to GPLv3 (but for the additional condition recited earlier). A more detailed description of AGPL and its effect on cloud computing and SaaS can be found in Chapter 17.

3.2.2.1.4 Other copyleft licences: MPL and EPL

Although there are but a few other Open Source licences that attempt to create a strong copyleft effect, there are two very prominent other Open Source licences that create a weak copyleft effect: The Mozilla Public License (MPL)[93] and the Eclipse Public License (EPL).[94] Both of these licences are an effort—similar to the effort made with the Apache 2.0 license—to address some perceived ambiguities or missing features in LGPL, as well as to provide a clearer and more easy-to-use definition of the circumstances when non-copyleft code can make use of code licensed under those licences.

The MPL exists in two versions—MPLv1.1[95] and MPLv2.0.[96] Both are weak copyleft licences written by the Mozilla Foundation, with version 2.0 being an

[91] GNU Operating System, 'GNU Affero General Public License, Version 3', § 13, <https://www.gnu.org/licenses/agpl-3.0.en.html> accessed 19 November 2007.

[92] Software Package Data Exchange (SPDX), 'Affero General Public License v1.0 only' <https://spdx.org/licenses/AGPL-1.0-only.html> accessed 11 March 2020.

[93] Mozilla Foundation, 'Mozilla Public License' <https://www.mozilla.org/en-US/MPL/> accessed 11 March 2020.

[94] Eclipse Foundation, 'Eclipse Public License—v 2.0' <https://www.eclipse.org/legal/epl-2.0/> accessed 11March 2020.

[95] Mozilla Foundation, 'Mozilla Public License Version 1.1' <https://www.mozilla.org/en-US/MPL/1.1/> accessed 11 March 2020 (MPLv1.0).

[96] Mozilla Foundation, 'Mozilla Public License Version 2.0' <https://www.mozilla.org/en-US/MPL/2.0/> accessed 11 March 2020 (MPLv2.0).

updated version having improved terminology. Like the Apache 2.0 license versus the MIT and BSD licenses, MPL attempts to improve upon issues in LGPL—for example, by addressing patent rights via an express patent grant, including defensive termination conditions for that grant, and by having more robust definitions and terms. The MPL licences are also an attempt to make a much clearer, and easier to understand and use in practice, distinction between code that must be copyleft and code that need not be. This distinction is set forth in the definitions of 'Modifications' to the 'Covered Software' provided under MPL:[97]

> '"Modifications" means any of the following: any file in Source Code Form that results from an addition to, deletion from, or modification of the contents of Covered Software; or any new file in Source Code Form that contains any Covered Software.' MPL also make clear that putting together such files into a 'Larger Work' does not subject the entire result to the MPL: '"Larger Work" means a work that combines Covered Software with other material, in a separate file or files, that is not Covered Software.[98] . . . You may create and distribute a Larger Work under terms of Your choice, provided that You also comply with the requirements of this Licence for the Covered Software.'[99]

Although the file-based distinction between that which must be MPL and that which need not is likely to be much easier to navigate than the exceptions to copyleft in LGPL, there is one potential ambiguity that bears a cautious approach: when creating a new file, designed to not be subject to MPL but intended to work with it, MPL states that 'Covered Software' is defined as 'Source Code Form to which the initial Contributor has attached the notice in Exhibit A, the Executable Form of such Source Code Form, and Modifications of such Source Code Form, in each case including portions thereof'.[100] Note the degree of recursion between the definition of 'Covered Code' and 'Modifications', where each includes reference to the other. A file that is separate from Covered Code, but which may need to reproduce elements (such as interfaces) in order to work with that Covered Code, could be argued to thus fall within the obligation to license that file only under MPL and to produce source code for that file. The Mozilla Foundation attempts to clarify this issue in its FAQs:

> Q11: . . . If I use MPL-licensed code in my proprietary application, will I have to give all the source code away?
>
> No. The license requires that Modifications (as defined in Section 1.10 of the license) must be licensed under the MPL and made available to anyone to whom you distribute the Source Code. However, new files containing no MPL-licensed

[97] MPLv2.0, § 1.10.
[98] MPLv2.0, § 1.7.
[99] MPLv2.0, § 3.3.
[100] MPLv2.0, § 1.4.

> code are not Modifications, and therefore do not need to be distributed under the terms of the MPL, even if you create a Larger Work .... *This allows, for example, programs using MPL-licensed code to be statically linked to and distributed as part of a larger proprietary piece of software, which would not generally be possible under the terms of stronger copyleft licenses.*[101]

Given the statement about static linking in the MPL FAQs, it is certainly also the case that dynamic linking of these files would not cause the MPL to attach to the file so linked. The extent the FAQs would be dispositive on the issue of linking of MPL and non-MPL files has yet to be determined, although the fact that the licence steward, the Mozilla Foundation, accepts this distinction is likely to be found highly persuasive.

The EPL also exists in two versions—EPLv1.0[102] and EPLv2.0.[103] Both are weak copyleft licences written by the Eclipse Foundation, with version 2.0 being an updated version having improved terminology. Like MPL, EPL attempts to improve upon issues in LGPLv2.1: patent rights, patent defensive termination, and more robust definitions and terms. EPLv2.0 has a slight advantage over MPLv2.0 in that it removes from the terms of the licence the ambiguity around including interfacing code in order to allow non-EPL-licensed code to interoperate with EPL-licensed code:[104]

> 'Modified Works' shall mean any work in Source Code or other form that results from an addition to, deletion from, or modification of the contents of the Program, including, for purposes of clarity any new file in Source Code form that contains any contents of the Program. *Modified Works shall not include works that contain only declarations, interfaces, types, classes, structures, or files of the Program solely in each case in order to link to, bind by name, or subclass the Program or Modified Works thereof.*[105]

[101] Mozilla Foundation, 'MPL 2.0 FAQ' <https://www.mozilla.org/en-US/MPL/2.0/FAQ/> accessed 2 March 2020 (emphasis added).

[102] Eclipse Foundation, 'Eclipse Public License—v 1.0' <https://www.eclipse.org/legal/epl-v10.html> accessed 11 March 2020.

[103] Eclipse Foundation, 'Eclipse Public License—v 2.0' <https://www.eclipse.org/legal/epl-2.0/> accessed 11 March 2020 (EPLv2).

[104] The ambiguity present in MPLv2.0 also existed in EPLv1.0, which defined a 'Contribution' not covered by the licence as 'additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own licence agreement, *and (ii) are not derivative works of the Program*' (emphasis added). The reference back to the idea of derivative works begged the question of whether the use of interfacing code that might legally result in the creation of a derivative work would curtail the exception and make EPLv1.0 closer to a strong copyleft licence. See Katie Osborne, 'License Profile: The Eclipse Public License' (2015) 1(1) *Journal of Open Law, Technology & Society* 1, at 6 <https://jolts.world/index.php/jolts/article/view/73/211>.

[105] EPLv2.0, § 1 (emphasis added).

In this way, EPLv2.0 makes much clearer the distinction between EPL and non-EPL files and acknowledges that in order for such files to work together there may need to be reproduction of certain interfacing code from the EPL files. The definition of 'Modified Works' in EPLv2.0 thus acknowledges that such reproduction does not spread the licence to that file.

3.2.2.1.5 'Weak copyleft' exception practice

In certain circumstances, authors have chosen to license their code under a strong copyleft licence but have attempted to provide a particular weak copyleft effect to enable certain uses of the code without imposing strong copyleft requirements on that code. Two of the most notable examples of this is the so-called sys call exception that is used with the Linux kernel (part of the GNU/Linux operating system) and the 'runtime' exception used with the GNU C++ Compiler (GCC). In each case, a version of GPL is used for the base code licence, but the authors have included a notice (usually included immediately above the licence text, or in a 'README' file appended to the source code) to spell out that for certain uses, GPL would not apply.

The Linux kernel, licensed under GPLv2, includes the following exception (often called the 'sys call' or 'system call' exception), authored by the original author of that kernel, Linus Torvalds:

> NOTE! This copyright does *not* cover user programs that use kernel services by normal system calls—this is merely considered normal use of the kernel, and does *not* fall under the heading of 'derived work'. Also note that the GPL below is copyrighted by the Free Software Foundation, but the instance of code that it refers to (the Linux kernel) is copyrighted by me and others who actually wrote it.[106]

In this way, the authors of the Linux kernel attempt to weaken some of the effect of GPLv2 against their authored code by making clear that as authors, and therefore granters of the licence, certain uses that other code might make of facilities in the kernel, even if in the law of a relevant jurisdiction might be considered a derivative work subject to the requirements of GPLv2, are not considered so by those authors. In this way, for this particular code, GPLv2 is turned into a weak copyleft licence, but not using the definitions of exceptions to the licence that are used in, *inter alia*, LGPL.

The GCC also uses an exception to specify that certain uses of the code from that tool—licensed under GPLv3—do not spread the licence attached to that code.[107]

[106] <https://github.com/torvalds/linux/blob/master/LICENSES/exceptions/Linux-syscall-note> accessed 13 April 2022.

[107] GNU Operating System, 'GCC Runtime Library Exception version 3.1' (31 March 2009) <https://www.gnu.org/licenses/gcc-exception-3.1.en.html> accessed 13 April 2022.

The wording of the exception itself is somewhat complex, but the intent of the exception is described before the text of that exception:

> When you use GCC to compile a program, GCC may combine portions of certain GCC header files and runtime libraries with the compiled program. The purpose of this Exception is to allow compilation of non-GPL (including proprietary) programs to use, in this way, the header files and runtime libraries covered by this Exception.[108]

The purpose of this exception is to prevent mere use of the tool to compile code from causing the resulting compiled code to be GPL—thus limiting the use of the tool itself to only GPLv3 licensed programs.[109]

There are numerous other examples of software licensed under an Open Source licence where the authors have granted some form of exception where certain uses of their code are not obliged to comply with all or part of the terms of that licence; it is good practice, when confronted with situations where a proposed use of Open Source licensed software might result in a less than desirable licensing outcome (either a licence conflict, or the requirement to use a licence that may be undesirable in a particular usage case) to examine the source code repository to see if an author exception has been granted.

3.3 Software Interaction and Licence Compatibility

3.3.1 The linking question

As described in section 3.1.4.1, linking—either statically, before run-time, or dynamically, during run-time—is a relatively common programing technique to allow two or more software programs or modules to operate together or to be combined. There has long been a debate about the effect of linking on programs licensed using Open Source licences—particularly copyleft licences.[110] Some licences, like Apache, attempt to make clear that irrespective of whether the law in a particular jurisdiction would find a particular type of link to create a derivative work, certain forms of linking do not cause the licence to apply to the result. In that case, it is relatively safe to assume that such an exception would be found controlling on the question of applicability of the licence. Other licences, like GPLv3, attempt to make clear that certain types of linking should be considered to create a derivative

[108] 'GCC Runtime Library Exception version 3.1' see note 111.

[109] GNU Operating System, 'GCC Runtime Library Exception Rationale and FAQ' <https://www.gnu.org/licenses/gcc-exception-3.1-faq.html> accessed 11 March 2020.

[110] Bain, 'Software Interactions and the GNU General Public License', see note 83, at 177.

work (or at least, to be governed by the terms of the licence) and therefore cause the licence to apply to the result. Whether the courts in a particular jurisdiction would find such a statement of intent applicable is an as-yet unresolved issue.

There is also an important issue concerning the particular jurisdiction in which the linking question is confronted. The term 'derivative work' has a statutory meaning under US law, although it is subject to different interpretive tests at the present, as discussed in more detail in section 3.1.4.2 earlier in this chapter. In other jurisdictions such as those in the UK and EU, the analysis is not even that clear-cut, due to the lack of definition of the term.

3.3.1.1 Other interaction issues: technical impediments

The Linux kernel, licensed under GPLv2 only, instituted—in around 2002—a technical impediment intended to discourage the loading of LKMs at run-time which were not licensed under GPLv2 or a GPLv2 compatible licence.[111] This technical impediment issues an error message ('Kernel is Tainted for following reasons: Proprietary module was loaded') when such a non-GPLv2 compatibly licensed LKM is loaded by the kernel.[112] In general, the community of Linux kernel developers and maintainers believe that run-time LKMs that are 'proprietary' (typically closed source, but this impediment would also display a message for LKMs licensed under a non-GPLv2-compatible licence) should not be allowed, and this technical impediment was intended to discourage the creation or use of such LKMs.[113] There have been efforts to circumvent this technical impediment in order to allow the loading of proprietary LKMs without the display of the warning message, although the reaction by the kernel maintainer community to such efforts is decidedly negative if not hostile.[114] Avoiding such impediments to allow proprietary LKMs to load may also potentially run afoul of laws—like the DMCA in the US—intended to prevent circumvention of technological impediments by way of DRM.

3.3.2 Specific compatibility issues

Navigating which of the many Open Source licences are compatible with one another, and under what technical conditions and in which legal jurisdictions, is a highly complex issue. The answer will depend at least in part on how two or more pieces of software interact with one another, the extent to which interfaces or other

[111] Linux Kernel Mailing List (LKML), 'The tainted message' (26 April 2002), <http://lkml.iu.edu/hypermail/linux/kernel/0204.3/0428.html> accessed 13 April 2022.

[112] Linux Kernel, 'The Linux kernel user's and administrator's guide: Tainted kernels', <https://www.kernel.org/doc/html/latest/admin-guide/tainted-kernels.html> accessed 11 March 2020.

[113] Linux.com, 'Tainted love: proprietary drivers and the Linux kernel' (28 April 2004), <https://www.linux.com/news/tainted-love-proprietary-drivers-and-linux-kernel/> accessed 13 April 2022.

[114] LKML, 'Linuxant/Conexant HSF/HCF modem drivers unlocked' (29 October 2004) <http://lkml.iu.edu/hypermail/linux/kernel/0410.3/2190.html> accessed 13 April 2022.

code may need to be reproduced to facilitate that interaction, the specific wording of the licences involved, and whether there are any exceptions granted by the authors that might contemplate and except out from general licence conditions certain types of interactions. The discussion which follows attempts to summarise these compatibility issues, but practitioners are cautioned that there are many different factors that need to be considered, and the law on, *inter alia*, copyrightability, functional dictation/merger doctrine, and fair dealing/fair use continue to evolve, and cases still being considered as of publication may significantly impact questions related to compatibility.

Table 3.2 is a graphical representation intended to summarise how the licences discussed earlier are, are not, or may possibly be, compatible. A single-headed arrow represents 'one way' compatibility—that is the licence at the start of the arrow is compatible with the licence at the end of the arrow, but not vice versa; and a double-headed arrow represents 'two way' compatibility, that is the licences are compatible in either direction. A solid line means compatibility in all circumstances, a dashed line means compatibility in certain circumstances, and a dotted line means there is an unresolved debate about whether compatibility exists. An 'X' means there is no compatibility in either direction. Note that this chart does not take into account that there may be some degree of compatibility in certain specific

Table 3.2 Compatibility between Certain Open Source Licences

	GPLv2								
GPLv3	X	GPLv3							
AGPLv3	X	——↑	AGPLv3						
LGPLv2.1	——↑	——↑	X	LGPLv2.1					
LGPLv3.0	X	——↑	X	← –↑	LGPLv3.0				
Mozilla	——↑	——↑	——↑	——↑	——↑	Mozilla			
Eclipse	X	X	X	← –↑	← –↑	← –↑	Eclipse		
BSD	——↑	——↑	——↑	——↑	——↑	——↑	——↑	BSD	
MIT	——↑	——↑	——↑	——↑	——↑	——↑	——↑	←—↑	MIT
Apache 2.0	– –↑	——↑	——↑	– –↑	——↑	——↑	——↑	←—↑	←—↑

X = Two-way incompatibility

——↑ = One-way compatibility (in direction of arrow)

– –↑ = One-way partial compatibility (in direction of arrow)

←—↑ = Two-way compatibility

← –↑ = Two-way partial compatibility

circumstances as a result of author exceptions, as discussed in section 3.2.2.1.5 earlier, or licence statements that allow the user to receive the code under later versions of the licence, as discussed in section 3.3.2.1 which follows.

3.3.2.1 'Strong' copyleft licences

Those licences which are generally referred to as having 'strong' copyleft provisions have the hallmark of imposing their terms on any downstream exercise of the right to make modifications (or in the case of the US, create 'derivative works'). Because of this, attempting to interoperate 'strong' copyleft licensed code with code that has any terms that might conflict with the terms in the 'strong' copyleft licence, presents potential compatibility problems that puts the interoperating code at risk of violating the 'strong' copyleft licence's terms (as well as the terms for the interoperating code). In this way, the strong copyleft licences (GPLv2, GPLv3, AGPL) are generally incompatible with one another. There are a few exceptions to this general rule. First, there has been a practice for a number of authors of code licensed under GPLv2 to include a statement that the code is licensed under 'GPLv2 or any later version'.[115] Code licensed under this form of a GPLv2 licence notice is compatible with GPLv3; otherwise, code licensed under GPLv2 only (which is the case with the Linux kernel)[116] is incompatible with GPLv3, LGPLv3, and AGPLv3.[117] Second, the newer versions of the GNU family of licences were designed to allow a certain degree of one-way compatibility with one another; thus, LGPLv3 may be combined with GPLv3 as long as the resulting combination is licensed GPLv3;[118] similarly, AGPLv3 includes a provision that allows combinations with GPLv3, as long as the resulting combination is licensed GPLv3.[119] Third, LGPLv2.1 includes a section stating that 'You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy' of code licensed under LGPLv2.1;[120] thus LGPLv2.1 is one-way compatible with GPLv2 and GPLv3.

3.3.2.2 'Weak' copyleft licences

The weak copyleft licences tend to have limited compatibility. A weak copyleft licence typically stipulates that derivative content must be licensed under that same licence, but admits of articulated exceptions to that general rule.

[115] <https://spdx.org/licenses/GPL-2.0-or-later.html> accessed 13 April 2022.

[116] Kernel.org, 'Working with the kernel development community: Linux kernel licensing rules' <https://www.kernel.org/doc/html/latest/process/license-rules.html> accessed 11 March 2020.

[117] Free Software Foundation, 'Frequently Asked Questions about the GNU Licenses' <https://www.gnu.org/licenses/gpl-faq.en.html#AllCompatibility> accessed 3 March 2020 (the chart does not address AGPLv3, but by the logic of GPLv3 incompatibility, AGPLv3 would be equally incompatible).

[118] FSF, 'Frequently Asked Questions about the GNU Licenses' see note 121.

[119] AGPLv3, § 13.

[120] LGPLv2.1, § 3.

The MPL includes provisions, Sections 1.12 and 3.3, designed to allow combinations with the GNU family of licences, such that combinations of the two would be governed by the terms of the GNU-family licence. Thus, MPL is one-way compatible with GPLv2.1, GPLv3, LGPLv2.1, LGPLv3, and AGPLv3.[121] With regard to the EPL, depending on which version of MPL and EPL, and the way in which code under MPL and EPL are designed to interoperate, there is the potential for the licences to be compatible; this is primarily an issue of whether the code is maintained in separate files, as discussed in more detail in section 3.2.2.1.4 earlier in this chapter.

The EPL, in contrast, does not include a provision designed to allow compatibility with the GNU family of licences; thus the EPL is incompatible with all of GPLv2.1, GPLv3, and AGPLv3.[122] With respect to LGPLv2.1 and LGPLv3, it is possible to construct an interaction between code under EPL and one of the LGPL licences, as long as doing so falls within the exceptions in both licences discussed in more detailed in sections 3.2.2.1.2 and 3.2.2.1.4 earlier.

Finally, LGPLv2.1 would normally be considered incompatible with LGPLv3, for many of the same reasons that GPLv2 is considered incompatible with GPLv3. However, LGPLv2.1 includes a provision that allows re-licensing of LGPLv2.1 code under GPLv2 or any later version of that licence.[123] LGPLv3 includes a similar provision.[124] Thus, LGPLv2.1 and LGPLv3 can be made compatible, but the result would be licensing under GPLv3.[125]

3.3.2.3 MIT and BSD

Few complications arise with respect to compatibility between and with the permissive licences; indeed, it is a feature of these licences to provide broad compatibility. Copying and linking (which would appear to fall within the broad grant of rights in these licences) are broadly permitted by MIT and BSD, with only minimal requirements. Thus, as shown in Table 3.2, both of these licences are one-way compatible with all the Open Source licences discussed above, and are two-way compatible with each other and with Apache.

3.3.2.4 Apache

The Apache License also has a goal of being highly permissive and broadly compatible, in the same way as MIT and BSD. In most instances, as shown in Table 2 above,

[121] Free Software Foundation, 'Various licenses and comments about them' <https://www.gnu.org/licenses/license-list.html#MPL-2.0> accessed 11 March 2020.

[122] Free Software Foundation, 'Various licenses and comments about them' <https://www.gnu.org/licenses/license-list.html#EPL> accessed 11 March 2020.

[123] LGPLv2.1, § 3.

[124] LGPLv3, § 2b.

[125] Free Software Foundation, 'Frequently Asked Questions about the GNU licenses' <https://www.gnu.org/licenses/gpl-faq.en.html#AllCompatibility> accessed 11 March 2020.

the Apache License achieves that goal—being one-way compatible with most of the licences discussed above, and two-way compatible with MIT and BSD.

The Apache License does present a complication with regard to the GNU family of licences. The FSF has stated that the Apache 2.0 license is incompatible with GPLv2.[126]Although the ASF disagrees with this assessment, it nevertheless states that with regard to use of GPLv2, 'you should always try to obey the constraints expressed by the copyright holder when redistributing their work'.[127] This incompatibility introduces certain complications in architecting software stacks that may include Apache 2.0 and GPLv2 code (in particular, the Linux kernel of the GNU/ Linux operating system, which is licensed under GPLv2 only), and could very well be the reason why the more recent, and more carefully drafted, Apache 2.0 license has not supplanted the continued popularity of the BSD and MIT licences as *de facto* choice when selecting a permissive licence. Note that this incompatibility does not exist for the later generations of the GNU family of licences—GPLv3, LGPLv3, and AGPLv3—as it was an express goal during the process of updating those licences to allow them to be Apache License one-way compatible.[128] Thus, as reflected in Table 3.2, the Apache License is arguably one-way compatible with GPLv2, partially one-way compatible with LGPLv2.1, and fully one-way compatible with GPLv3, LGPLv3, and AGPLv3.

3.4 Interpreting Open Source Licences: Contract or 'Bare Licence'?

Although a discussion of specific enforcement cases and issues are described and analysed in detail in Chapter 5, there has been a long-standing debate amongst Open Source licence drafters, users, and potential enforcers over the question of whether such licences operate as 'bare licences', or should be interpreted and enforced as contracts. This question can be thought of—until very recently—as significantly academic, and much of the debate has been in academic circles but is nonetheless important when a particular author is (i) choosing a licence for their work, or (ii) considering enforcing that licence against another person or entity which they believe to be failing to comply with its terms. Although there is as yet no definitive answer to this debate—and in fact, the answer may be dependent upon the particular licence being enforced, and possibly the jurisdiction in which

[126] See FSF Licence Comments, see note 59, <https://www.gnu.org/licenses/license-list.html#apache2>. Note that by the logic used in the FSF's commentary on the Apache 2.0 licence, Apache 2.0 would likely be also incompatible with the Affero GPLv1.0 licence, and partially incompatible with the Lesser GPLv2.1 licence.

[127] Apache Software Foundation, 'GPL compatibility' <https://www.apache.org/licenses/GPL-compatibility.html> accessed 10 March 2020.

[128] Brett Smith, 'A Quick Guide to GPLv3' <https://www.gnu.org/licenses/quick-guide-gplv3.html> accessed 11 March 2020.

enforcement is contemplated—the trend in adjudication would appear to be supportive of the bare licence theory of licence interpretation and enforcement, or indeed possibly that either theory could be pursued by an author of software licensed under an Open Source licence, in the event they desire to engage in licence enforcement.

3.4.1 Open Source licences as bare licences

It has long been the position of the FSF that the licences over which it exercises stewardship operate as 'bare licences'.[129] Under this theory, the author grants a unilateral permission, under the IP rights either expressly enumerated in the licence text, or impliedly granted as a result of the structure and text of the licence and conditions under which it was granted, to engage in activities the author would otherwise have exclusive rights to practice.

Thinking of Open Source licences—or at least those licences that do not explicitly present themselves as contracts between the author(s) and licensee(s)—as mere unilateral permission from the author to each particular user, provides some potentially advantageous benefits to the author(s). First, the necessity to worry about the fundamental requirements for establishing the existence of a contract—offer, acceptance, consideration, intent, certainty, and completeness[130]—need not be established in order to pursue a violator, nor may privity of contract with the particular violator need be established. Second, given the numerous variations in rules governing contract law (which, for example, are analysed state-by-state in the US), viewing an Open Source licence as merely a permission under certain IP rights which—if not followed—result in a claim for violation of those rights, may allow the application of more uniform law, and provide more flexibility and ease in pursuing remedies, than pursuing relief under contract.[131]

The trend of enforcement and judicial interpretation of Open Source software licences suggests that the bare licence theory of Open Source licence interpretation and enforcement is valid, and may be preferable to those authors wishing to exercise their right of enforcement against accused licence violators.

The *Jacobsen v Katzer*[132] case in the US led the way—at least in common law jurisdictions—in validating the bare licence theory of Open Source software

[129] See Eben Moglen and Richard Stallman, 'Transcript of Opening Session of the First International GPLv3 Conference', 16 January 2006 <http://www.ifso.ie/documents/gplv3-launch-2006-01-16.html> accessed 16 January 2020. Eben Moglen, 'Enforcing the GNU GPL' (10 September 2001), <https://www.gnu.org/philosophy/enforcing-gpl.html> accessed 28 February 2020.

[130] Catharine MacMillan and Richard Stone, 'Elements of the law of contract' (2012) University of London International Programmes, <https://www.dphu.org/uploads/attachements/books/books_4071_0.pdf> accessed 13 April 2022.

[131] See GPLv3 Transcript, note 28.

[132] *Jacobsen v Katzer,* 535 F.3d 1373 (Fed. Cir. 2008).

licensing. The *Jacobsen* case involved enforcement of the terms of the Artistic Licence,[133] a permissive licence that includes obligations to include attribution notices and to identify modifications made. The defendant used code licensed by the plaintiff under the Artistic License, but failed to provide attribution or identify modifications. According to the District Court in *Jacobsen*, defendant's violation of the requirements of the Artistic License constituted a breach of contract, rather than use of the plaintiff's copyrights outside of the conditions of the licence and thus copyright infringement.[134] The Court of Appeals for the Federal Circuit overruled the District Court's ruling, holding that the requirements of Artistic License were not independent contractual covenants but merely conditions attached to the copyright grant. Because the defendant's actions had gone beyond the scope of the licence—by failing to comply with the fairly minimal conditions of the Artistic License—an action for copyright infringement could be brought by the author, and remedies for copyright infringement could be sought.[135]

Given the similarity in the grants and conditions between the Artistic License and the BSD and MIT Licenses, it would seem likely that at least those licences would also be interpreted to operate as bare licences, at least in the US, under the reasoning of the *Jacobsen v Katzer* decision. As noted earlier, the FSF—stewards of the GNU family of licences—has long advocated that those licences are also bare licences and not contracts, and commentators have acknowledged that that may be a viable interpretation of those licences.[136] Although there is no clear UK case regarding the bare licence versus contract issue concerning Open Source licensing, some commentators believe the rationale of *Jacbosen v Katzer* would equally apply there.[137]

Civil law jurisdictions also appear to generally accept the bare licence theory as at least one way to interpret Open Source licences.[138] A recent decision of the ECJ, upon appeal of a decision emanating from France, seems to bear out the theory

[133] Ironically, given that this case is perhaps the most consequential decisions interpreting the obligations of an Open Source licence, it involves one of the least popular Open Source licences. *See* Ben Balter, 'Open Source License Usage of GitHub.com' *The GitHub Blog* (9 March 2015), <https://github.blog/2015-03-09-open-source-license-usage-on-github-com/> accessed 3 February 2021 (showing the Artistic Licence as the seventh most used of sixteen open source licences on GitHub).

[134] *Jacobsen v Katzer*, No. 06-CV-01905 JSW, 2007 WL 2358628 (N.D.Cal. 17 August 2007).

[135] *Jacobsen v Katzer*, 535 F.3d 1373 at 1381–3 (Fed. Cir. 2008).

[136] Mark Henley, 'Jacobsen v Katzer and Kamind Associates—An English Legal Perspective' (2009) 1(1) *Journal of Open Law, Technology and Society* 41, at 43 (2009); Noah Shemtov, 'FOSS License: Bare License or Contract', presentation available at <https://web.ua.es/es/contratos-id/documentos/itipupdate2011/shemtov.pdf> accessed 12 March 2020.

[137] Shemtov, 'FOSS License: Bare License or Contract', see note 140.

[138] German cases include *Welte v Sitecom Deutschland GmbH*, District Court of Munich, 19 May 2004, case 21 O 6123/04; *Welte v Skype Technologies S A*, District Court of Munich, 12 July 2007, case 7 O 5245/07. A French case, *EDU 4 v AFPA*, Cour d'Appel de Paris, Pole 5, Chambre 10, no: 294, discusses GPL although not in detail. See Martin von Willebrand, 'Case Law Report: A Look at EDU 4 v. AFPA, also Known as the "Paris GPL case"' (2009) 1(2) *Journal of Open Law, Technology and Society* 123, at 123–26.

that a copyright licence violation can indeed be pursued whether or not there may be a contractual basis for a claim against the licence violator:

> According to Article 2(1) of Directive 2004/48 [of the European Parliament and of the Council of 29 April 2004 on the enforcement of intellectual property rights], that directive applies to 'any infringement of intellectual property rights.' It is apparent from the wording of that provision, in particular from the adjective 'any', that that directive must be interpreted as also covering infringements resulting from the breach of a contractual clause relating to the exploitation of an intellectual property right, including that of an author of a computer program.[139]

3.4.2 Open Source licences as contracts

There is a relatively robust line of argument that Open Source licences—or at least a selected subset of Open Source licences—operate as valid contracts between the authors of the licensed code and the recipients of that code.[140] Others have argued to the contrary:

> A contract … is an exchange of obligations, either of promises for promises, or of promises of future performance, for present performance, or payment. The idea that 'licenses' to use patents or copyrights must be contracts is an artefact of twentieth-century practice, in which licensors offered an exchange of promises with users: 'We will give you a copy of our copyrighted work,' in essence, 'if you pay us and promise to enter into certain obligations concerning the work.'[141]

As discussed earlier, several of the more commonly-used Open Source licences are likely to be evaluated as bare licences—if the author chose to present that theory to a judicial tribunal—although the interpretive decisions validating that theory do not preclude an author from also pursuing a claim for breach of contract.

At least one court has interpreted GPLv2 under a contractual analysis and has rejected the application of a 'bare licence' theory for enforcement of that licence. In the French court decision in *Entre'Ouvert v Orange*,[142] the court—in

[139] *IT Development SAS v Free Mobile SAS*, ECLI:EU:C:2019:1099 (Fifth Chamber, CJEU, 18 December 2019). Compare that decision to the outcome in *Entre'Overt v Orange*, Tribunal de grande instance (TGI) of Paris, 3rd chamber, 3rd section (21 June 2019), discussed in section 3.4.2 below.

[140] See Robert W Gomulkiewicz, 'How Copyleft Uses License Rights to Succeed in the Open Source Software Revolution and the Implications for Article 2B' (1999) 36 *Houston Law Review* 179, at 194; Rosen, 'Open Source Licensing', see note 67, at 57–66.

[141] Eben Moglen, quoted in Pamela Jones, 'The GPL is a License, Not a Contract' lwn.net (3 December 2003) <https://lwn.net/Articles/61292/> accessed 2 March 2020.

[142] *Entre'Ouvert v Orange*, Tribunal de grande instance (TGI) of Paris, 3rd chamber, 3rd section (21 June 2019) <https://www.legalis.net/jurisprudences/tgi-de-paris-3eme-ch-3eme-section-jugement-du-21-juin-2019/> accessed 8 June 2022.

interpreting a claim of failure to follow the requirements of GPLv2—stated that because Entre'Ouvert was seeking compensation for damage caused by Orange's failure to perform obligations in GPLv2—specifically providing source code—the defendant was not operating outside of the GPLv2 license and as a result the only claim Entre'Ouvert could pursue was under French contract law, for a contractual breach of the requirements of GPLv2.[143] This decision would appear to be conflict with the ECJ's interpretation of French law in *IT Development SAS v Free Mobile SAS*,[144] despite the *Entre'Ouvert* decision being issued previously, but which was not cited in the *IT Development* decision. The extent to which this conflict will be resolved, and its effect on potential enforcement actions in France, have not yet been clearly established.

In 2021, a lawsuit was filed in the US with the intent of definitively establishing not only a contract theory for the GPL family of licences, but also to open up the possibility that recipients of Open Source—rather than just authors of Open Source—could enforce the terms of those licences. In *Software Freedom Conservancy, Inc. v Vizio, Inc.*,[145] a lawsuit was filed—in the state courts of California—to enforce GPLv2 based on allegations that 'complete corresponding source' had not been provided to purchasers of products sold by Vizio containing GPLv2 binaries.[146] The Software Freedom Conservancy, as one of the purchasers, asserted it was a third-party beneficiary of the contractual right in GPLv2 to receive source code, and therefore had the right to sue to enforce that licence.[147] In response, Vizio attempted to have that lawsuit 'removed' (transferred) from state court to US federal court, arguing that any violation of GPLv2 may only be pursued as a claim of copyright infringement, which are heard exclusively in US federal courts.[148] The federal court decided that violations of the obligation to provide source under GPLv2 could be pursued as a matter of contract law, and therefore US state courts could decide such claims under state contract law.[149] Thus, at least in the US, the potential for pursuing GPL violation claims as copyright infringements, in US federal courts, and as contract breaches, in US state courts, may be a possibility—depending on the eventual outcome of the *Vizio* litigation.

At least some Open Source licences intentionally present themselves as contracts as well as licences.[150] At a minimum, the most popular Open Source licences

[143] *Entre'Ouvert v Orange*, note 146.

[144] *IT Development SAS v Free Mobile SAS*, ECLI:EU:C:2019:1099 (Fifth Chamber, CJEU).

[145] Case No. 30-2021-01226723-CU-BC-CJC (Cal. Super. Ct., Orange County, filed 19 October 2021) (*Vizio* state case).

[146] *Vizio* state case, note 149, Complaint at paras 48–77.

[147] *Vizio* state case, note 149, Complaint at paras 87–126.

[148] *Software Freedom Conservancy, Inc. v Vizio, Inc.* Case No. 8:21-cv-01943, Notice of Removal of Action to Federal Court (C. D. Cal. 29 November 2021) (*Vizio* federal case).

[149] *Vizio* federal case, note 152, Order Granting Plaintiff's Motion for Remand (C. D. Cal. 13 May 2022).

[150] Rosen, 'Open Source Licensing', see note 67, at 59 (discussing the Academic Free License and the Open Software License).

do not explicitly preclude an interpretation that they could be enforced as bare licences, but only as contracts. Given the weight of both enforcement theories, and judicial decisions to date, it seems clear that the contract theory could remain a minority theory of Open Source software licensing interpretation, and is unlikely to a majority theory of license enforcement absent a significant decision calling into question the application of the bare licence theory to particular licences or in particular jurisdictions.

3.5 What Makes a Software Licence 'free' or 'open source'?

Although this volume refers to 'free and open source licences collectively as open source, that use does not necessarily represent a unitary concept. 'Free and open source' licensing actually represent two different, but significantly coextensive, classes of licences: 'free' software licences, and 'open source' software licences.

3.5.1 Free software licences

The class of 'free' software licences is typically recognised as those licences that meet the Free Software Definition (FSD),[151] as maintained by the FSF, and that have been validated by the FSF and added to their list of free software licences.[152] The FSD is a four-part test against which licences are measured to determine if they promote the FSF's concept of 'software freedom':

- The freedom to run the program as you wish, for any purpose (freedom 0).
- The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help others (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.[153]

The FSF's 'four freedoms' are the minimum standards necessary for a particular software licence to be considered a 'free software' licence; as the FSF states:

[151] GNU Operating System, 'What is free software? The Free Software Definition' <https://www.gnu.org/philosophy/free-sw.html.en> accessed 4 February 2021.

[152] GNU Operating System, 'Various licenses and comments about them: software licenses' <https://www.gnu.org/licenses/license-list.html#SoftwareLicenses> accessed 4 February 2021.

[153] FSD, see note 146.

> [C]riteria such as those stated in this free software definition require careful thought for their interpretation. To decide whether a specific software license qualifies as a free software license, we judge it based on these criteria to determine whether it fits their spirit as well as the precise words. If a license includes unconscionable restrictions, we reject it, even if we did not anticipate the issue in these criteria. Sometimes a license requirement raises an issue that calls for extensive thought, including discussions with a lawyer, before we can decide if the requirement is acceptable. When we reach a conclusion about a new issue, we often update these criteria to make it easier to see why certain licenses do or don't qualify.[154]

The FSF maintains an extensive list of software licences that it has determined, based on the criteria discussed earlier, meet its standards in order to qualify as a 'free software' licence, as well as a list of licences which it has determined are 'non-free' because they fail these criteria.[155] The 'free software' licences are subdivided in two categories: 'free software' licence that are 'GPL-compatible' and those that are 'GPL-incompatible'.[156] The measure of GPL compatibility is determined by evaluating whether the licence in question is one-way compatible with GPL.[157] GPL compatibility is an important criterion for the FSF, as the FSF promotes the GPL as the optimal licence for software freedom.[158]

There is no formal process for validating that a licence is a 'free software' licence and therefore to add a licence to the FSF's list of 'free software' licence; licences may be submitted via email to the FSF, but there is no formal review process or timeline specified by the FSF for making such decisions or adding licences to its lists.[159]

3.5.2 Open source software licences

The class of 'open source' software licences is recognised as those licences that meet the Open Source Definition (OSD), as maintained by the OSI, and that have been validated by an approval process run by the OSI. The OSD is a ten-part test against which licences are measured to determine if are 'open source':

1. Free Redistribution
   The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing

[154] FSD, see note 146.

[155] FSF Software License List, see note 147.

[156] FSF Software License List, see note 147.

[157] GNU Project, 'What does it mean to say a license is "compatible with the GPL?"' <https://www.gnu.org/licenses/gpl-faq.html#WhatDoesCompatMean> accessed 5 February 2021.

[158] GNU Project, 'Why you shouldn't use the Lesser GPL for your next library' <https://www.gnu.org/licenses/why-not-lgpl.html> accessed 5 February 2021.

[159] FSF Software License List, see note 147.

> programs from several different sources. The license shall not require a royalty or other fee for such sale.
>
> 2. Source Code
>   The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost, preferably downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.
> 3. Derived Works
>   The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
> 4. Integrity of The Author's Source Code
>   … The license must explicitly permit distribution of software built from modified source code.…
> 5. No Discrimination Against Persons or Groups
>   The license must not discriminate against any person or group of persons.
> 6. No Discrimination Against Fields of Endeavor
>   The license must not restrict anyone from making use of the program in a specific field of endeavor.…
> 7. Distribution of License
>   The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
> 8. License Must Not Be Specific to a Product
>   The rights attached to the program must not depend on the program's being part of a particular software distribution.…
> 9. License Must Not Restrict Other Software
>   The license must not place restrictions on other software that is distributed along with the licensed software.…
> 10. License Must Be Technology-Neutral
>   No provision of the license may be predicated on any individual technology or style of interface.[160]

The OSI maintains an extensive list of software licences that it has determined, based on the criteria discussed earlier, meet its standards in order to qualify as an

[160] Open Source Initiative, 'The Open Source Definition' <https://opensource.org/osd> accessed 5 February 2021.

'open source' licence.[161] The list is sorted into one sub categorisation, established by the OSI, so as to identify licences that are 'are popular, widely used, or have strong communities', as well as other subcategories of licences that do not meet that test.[162] The 'popular, widely used, or have strong communities' licences as identified by the OSI are:

- Apache License 2.0
- BSD 3-Clause and BSD 2-Clause Licenses
- All versions of GPL
- All versions of LGPL
- MIT License
- Mozilla Public License 2.0
- Common Development and Distribution License (CDDL)
- Eclipse Public License version 2.0

Unlike the 'free software' licence list maintained by the FSF, the OSI has a formal, documented process for submitting, evaluating, and approving licences to add to the list of 'open source' licences.[163] Licence submitters are requested to provide the following information concerning the licence for which they request OSI approval:

- Rationale: Clearly state rationale for a new license
- Distinguish: Compare to and contrast with the most similar OSI-approved license(s)
- Legal review: Describe any legal review the license has been through, and provide results of any legal analysis if available
- Proliferation category: Recommend which license proliferation category is appropriate[164]

The process itself is administered using a mailing list through which OSI members may submit comments, criticisms, or suggested changes regarding submitted licences, after which the OSI board conducts a vote as to whether a submitted licence should be added to the OSI 'open source' licence list, or other alternative actions are taken:

[161] Open Source Initiative, '[Open Source] Licenses by Name' <https://opensource.org/licenses/alphabetical> accessed 5 February 2021.

[162] Open Source Initiative, '[Open Source] Licenses by Category' <https://opensource.org/licenses> accessed 5 February 2021.

[163] Open Source Initiative, 'The License Review Process' <https://opensource.org/approval> accessed 5 February 2021.

[164] OSI License Review Process, see note 158.

- Defer for another 30-day discussion cycle, if community discussion of conformance of the license to the OSD remains active.
- Approve if, after taking into consideration community discussion, the OSI determines that the license conforms to the OSD and guarantees software freedom. A license may be approved on the condition that a change be made, but in general a license requiring changes will have to be resubmitted.
- Reject if (a) the OSI determines that the license cannot practically be remedied to adequately guarantee software freedom, or (b) there is sufficient consensus emerging from community discussion that the license should be rejected for substantive reasons, or (c) the license is problematic for non-substantive reasons (for example, it is poorly drafted or significantly duplicative of one or more existing OSI-approved licenses).
- Withhold approval, if (a) the OSI determines that approval would require reworking the license and (b) the license submitter appears willing and able to revise the license constructively.[165]

Although the OSI licence review process is open and relatively transparent, it has not been without controversy in some circumstances, and submitters have withdrawn licences from approval that they maintain are conformant with the OSD but which have encountered opposition during the approval process.[166]

3.6 Conclusion

Although the applicability of copyright to software—in any form that it may take—is by now well-established, and despite the over thirty-year history of Open Source licensing, there remain many unresolved questions about how software copyright should be analysed legally, or how certain aspects of Open Source software licences would be found to operate, if put to the test via court or other challenges.

Chapter 5 addresses the enforcement cases that have been pursued, and this chapter gives an overview of that which is—at the present—known about the operation of Open Source licences alone, or together with other licences.

Practitioners attempting to give advice on complex questions of licence interpretation in view of particular software programing and architectural scenarios should be cautious to appreciate that many questions remain unanswered, and the views expressed by many commentators can diverge, sometime quite radically. In addition, the answer to certain thorny questions about, for example, under

[165] OSI License Review Process, see note 158.

[166] Elliot Horowitz, 'Approval: Server Side Public License, Version 2 (SSPL v2)' (9 March 2019) <http://lists.opensource.org/pipermail/license-review_lists.opensource.org/2019-March/003989.html> accessed 5 February 2021.

what conditions a copyleft licence imposes its obligations on other software may be highly dependent upon the exact wording of that licence, the general state of copyright law interpretation both internationally, and nationally in the particular jurisdiction in which the software is being used, and the extent to which a deciding or interpreting body views the terms of that licence as the mere grant of enumerated copyright rights subject to certain conditions, or a reciprocal licence between the authors of the copyleft licensed code and the authors of the other software code. There are a handful of decided cases, in both the US and EU, which give some general guidance about how these issues might be resolved, but there is much room for additional decisions which could very well upend the way that Open Source licences are interpreted, and used in practice, today.

4 Contributor Agreements

Jilayne Lovejoy

4.1 Project Licence Agreements

From a legal perspective, regardless of how the project is organised or governed, there are two licences or agreements for every Open Source project.

'Inbound' refers to the licence or agreement under which contributions are made to the Open Source project.

'Outbound' refers to the licence under which end-users use the Open Source project.

As used here, the terms inbound and outbound are relative to the Open Source project.

There may be other licences for other content related to the project like documentation or data, but for our purposes here, we are discussing the licences or agreements as related to the code.

Chapter 3 considered outbound Open Source licences in detail and further chapters will discuss details regarding compliance and other aspects of those licences. These outbound licences for Open Source projects are, ostensibly, what makes the software 'free and open'.

However, if there is more than one person or entity working on an Open Source project, then there must be some legal understanding as to the inbound agreement governing contributions to the project. Recall that code can be assumed to have copyright and that restricts the use of a work (code) unless explicit permission is given, which amounts to a licence to use the code whether in writing or not. No use can be made without a licence or a transfer of the right of the creator. Thus, in order for the open collaborative model to work, the copyrightable code contributions

Jilayne Lovejoy, *Contributor Agreements* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Jilayne Lovejoy 2022. DOI: 10.1093/oso/9780198862345.003.0004

made to the project must be accompanied by a licence or some other transfer of rights. Without this the project would not be able to use those contributions.

A successful Open Source project will have contributors who must have clarity as to what the project can do (or not do) with their contributions. Likewise, contributors to a project should have clarity as to what rights they are giving the project.

Choosing an inbound licence, is more than a legal exercise. The Open Source movement was born from developers and engineers, not lawyers. Successful Open Source software projects approach licensing decisions within the larger context of the values, perspectives, and common practices of their community.

4.2 Types of Inbound Agreements for Open Source Projects

4.2.1 Inbound = outbound

'Inbound=outbound' refers to when contributions are accepted by the project under the same licence as the project's outbound licence. The contributor grants a licence to the project and each recipient of their contribution to the project under the same terms and conditions as the licence which users receive the project.

Copyright ownership remains with each contributor as this is a licence not an assignment. This distinction is often confused and sometimes the document titled, 'contributor licence agreement' (CLA), may be used on documents which are in fact assignments and vice versa. On that basis, it is important to inspect the actual document terms closely. A licence means that the contributor retains the ownership of the code and has the ability to do as they wish with the same code outside of the project, including licensing the same contribution code to someone else under a different licence.

Using the same licence for inbound and outbound provides a clean and equal set of rights coming into the project and going out, removing the need to analyse differences between inbound and outbound rights for discrepancies or inconsistencies.

Although there are no reliable statistics on inbound licensing to Open Source projects currently available, inbound = outbound is considered by far the most widely used and was the default for early Open Source projects. The model was so ubiquitous that the need for it to be named did not arise and the term 'inbound = outbound' was not coined until around 2010, when Richard Fontana, legal counsel at Red Hat, began using it in presentations and articles.[1]

Around that time in Open Source's history Contributor Licence Agreements (CLA) became more widely used, usually by well-known foundations or corporations and the Open Source community suddenly discovered a need to

[1] <https://ref.fedorapeople.org/fontana-linuxcon.html> accessed 13 April 2022.

distinguish the newer approach (i.e., CLAs) from what had been the norm (inbound=outbound).

If a specific inbound licence is not identified, the general assumption is that contributions are made under the project's outbound licence for that contribution at the time it was made, but for a number of reasons it is better to be explicit.

Inbound = outbound is easiest for both projects and contributors. It is likely that most contributors will already be familiar with the outbound licence (especially if it is an Open Source Initiative (OSI) approved licence) and thus, understand the rights they are giving to the project without the need to consult a lawyer. This is especially helpful for individual developers who may not have access to a lawyer. It also makes the approval process easier for contributions from developers or entities with lawyers who need to review the inbound licence terms for contributions to Open Source projects made on behalf of the entity.

There is little to no administrative work beyond including a copy of the licence and indicating it applies for both inbound and outbound code. Unlike a contributor agreement or assignment, the inbound = outbound model does not require the project to have a single person or entity with which to execute the agreement and hold the intellectual property rights.

Many developers and advocates prefer the inbound = outbound licensing model due to the egalitarian nature of the arrangement via symmetrical rights for contributors and users. Federated copyright ownership across contributors generally means that there is no potential single point of failure in a centralised accumulation of copyright licences or ownership, as with other models discussed later in the chapter. Copyright ownership being distributed among the contributors means the project must adhere to the wishes of each of the contributors in terms of major changes. For example, the only way the project could change the outbound licence across all code is to get every contributor's permission to do so.

Whilst there are differences in joint copyright enforcement in different jurisdictions, only copyright holders in the code have the legal 'standing' to enforce their rights against an infringing third party. Licence enforcement in this environment is not left to a single legal entity.

Detractors of inbound = outbound equally argue the difficulty in changing a project licence as a downside of this model but this must be weighed against the value of equality and fairness in contributions.

The LLVM project which uses inbound = outbound, undertook a licence change which included creating a foundation in order to have an entity that could implement the legal agreement to instantiate the agreement to the change of licence with all the contributors.[2] Their revised developer policy acknowledges the challenge of this task and states the rationale as follows:

[2] <https://foundation.llvm.org/docs/relicensing/> accessed 13 April 2022.

> changing the LLVM license requires tracking down the contributors to LLVM and getting them to agree that a license change is acceptable for their contributions. We feel that a high burden for relicensing is good for the project, because contributors do not have to fear that their code will be used in a way with which they disagree.[3]

Similarly, VLC also undertook a similar re-licensing effort citing, in addition to other factors, author's rights under French law.[4]

The licence can be changed under the inbound = outbound model but inevitably requires more work which may be a deterrent to licence change and which many developers view as a good thing.

4.2.2 Developer's Certificate of Origin

Unlike the other inbound agreements discussed in this chapter, the Developer's Certificate of Origin (DCO) is neither a licence nor an assignment. It has no express language granting rights from the contributor to the project. In fact, the DCO refers to submitting the contribution 'under the Open Source licence indicated in the file'.[5] For this reason, the DCO is a compliment to the inbound = outbound licence model and is generally not compatible with a contributor licence or assignment agreement.

The DCO is a statement affirming the contributor owns or has proper rights to contribute the code to the project. It also includes (as of v1.1, created in 2005) an acknowledgement that the contribution is a public record and any personal information required for sign-off will be maintained and redistributed as consistent with the Open Source project and licence.[6] Each contributor indicates agreement to the DCO by 'signing off' with their name and email address in each commit message.

The DCO was originally created for the Linux kernel in 2004 in response to the SCO lawsuits. The SCO Group brought a series of lawsuits against Linux vendors and users in 2003–2004 claiming copyright infringement, among other claims, of certain copyrighted code from UNIX that was contributed to Linux. The details in terms of the copyright ownership, legal proceedings, and various lawsuits are quite lengthy, highly contested, and spanned a decade or more.

One impact was an analysis of where specific code came from in the Linux kernel. Current-day tracking of contributed code did not exist at this point. The

[3] <https://llvm.org/docs/DeveloperPolicy.html#copyright> accessed 30 June 2022.
[4] <https://lwn.net/Articles/525718/> accessed 13 April 2022.
[5] <https://developercertificate.org/> accessed 13 April 2022.
[6] <https://lwn.net/Articles/139916/> accessed 13 April 2022.

Linux kernel community sought to connect contributors to copyrighted contributions and also to communicate to Linux developers that they had some responsibility regarding where their contributed code came from.

The use of a CLA for the Linux kernel was discussed and rejected due to its administrative burden and potential interference with the values of freedom and individualism. The ideas underpinning the DCO was personal accountability and keeping the representation and sign-off within the source control system. This could then be used to track all contributions made by a specific person.

4.2.2.1 Why use it?

Open Source software developers largely like the developer-friendly DCO; agreement with the DCO is effected via a Git command.[7] It has since been adopted by many Open Source projects beyond Linux and is just about as low-friction as inbound = outbound but provides more assurance to the project owner by way of the representation each contributor makes as to the provenance of their contributions.

4.2.3 Contributor Licence Agreements

Some projects use a different licence for incoming contributions from their outbound licence referred to as a CLA. At the most basic level, a CLA is an inbound licence under which an Open Source project receives contributions. The distinction from inbound = outbound is that with a CLA, the inbound licence terms are *not* the same as the outbound licence. The grants under a CLA may vary from the outbound licence grant, but the inbound licence must be as broad or broader than the outbound.

As a CLA is a licence, copyright ownership remains with the author. Developer communities frequently misunderstand this and CLAs are frequently discussed as if they are assignments, a fact not made better by the confusion of these terms in some inbound documentation.

Unless explicitly agreed otherwise in the CLA, standing to enforce copyright infringement does not transfer to the project owner but remains with each copyright author/contributor.

The licence grant in a CLA usually has minimal or no conditions, which gives the project owner greater flexibility in the outbound licence of the project or a change to that. A broad grant to the project with no conditions means the project owner can licence the code out under different or multiple licences or change the outbound licence without needing the permission of other contributors. For example, Elasticsearch and Kibana changed the project outbound license from Apache-2.0,

[7] <https://git-scm.com/docs/git-commit#Documentation/git-commit.txt--s> accessed 13 April 2022.

an OSI-approved licence, to a choice of two more restrictive, non-Open Source licences in 2021 to the distastes of many contributors who had contributed their code to the Open Source projects.[8]

That being said, CLAs may also include obligations upon the project owner, such as requiring the outbound licence to be Open Source or articulating a licence change process. This may alleviate developer concerns that projects will later swap an Open Source project to a proprietary licence.

In order to effect a CLA where all contributors grant a licence to the project, there must be a central legal entity at the recipient project to execute the legal agreement and to hold the licence grants. This might be an individual, a non-profit entity (whether charitable or trade organisation) or a corporation. In contrast to the decentralised and egalitarian nature inbound = outbound, use of a CLA generally means a centralised entity holds greater rights and power over the project than its contributors.

CLAs first appeared in the early or mid-2000s at a point of more formal corporate involvement in Open Source software projects and movement.

In the early 2010s after much community discussion a trend of turning away from the use of CLAs evolved, as noted by several high-profile project's public announcements.[9]

Today's focus in this area relates to automating the signature, sometimes at the expense of a real discussion as to the purpose or efficacy of the CLA for the given project.

The most well-known, often copied, and perhaps the earliest CLA is from the ASF. As a result of its not-for-profit status and goal of maintaining Open Source software, certain aspects of its CLAs are not appropriate in the context of a corporate project owner.

Nevertheless, many corporations have used the Apache CLA, usually with changes ranging from minor to more extensive modifications. Such changes create a certain amount of overhead for contributors and their lawyers reviewing something that looks similar, but requiring a careful look to spot the differences.

Unlike Open Source licences generally, there is not a large, developed body of 'standard' CLAs. Although you may hear people refer to 'CLA' as if it is a consistent, defined term, it is important to remember that CLAs are specific to the project and vary in terms. One must be careful when making assumptions about how CLAs work or what specific grants they require from contributors.

Several projects have created a standardised approach as an effort to reduce overall friction in the use of CLAs. Project Harmony began in 2010 led by Canonical with the intention of creating a suit of standard contribution agreements and

[8] <https://www.elastic.co/pricing/faq/licensing> accessed 13 April 2022.

[9] <https://www.infoworld.com/article/2608020/red-hat--joyent--and-others-break-down-licensing-barriers.html> accessed 13 April 2022.

includes an online 'agreement selector' that helps build a CLA or a copyright assignment agreement based on answers.[10] Canonical adopted a CLA from this suite.

ContributorAgreements.org also has a similar build-an-agreement interface.[11] This also includes the Fiduciary Licence Agreement (FLA), a project sponsored by the Free Software Foundation (FSF) Europe which aims to create an agreement whereby copyright is concentrated in one entity and that entity is obligated to ensure the software remains free and open.[12]

These projects were collaborative efforts among Open Source legal experts. If a CLA must be used, these projects provide a good starting point by way of a community process with world-leading experts, that resulted in vetted drafting.

4.2.3.1 Why use it?

CLAs are sometimes preferred by company lawyers as they meet the company's standard documentation and as the project owner, it allows them to more easily use a different outbound licence.

CLAs need to be drafted and implemented in a way to allow this goal. In order for the project owner to ensure the ability to change or use a different outbound licence, a CLA must be obtained from every contributor who has contributed code to the project that surpasses the threshold of copyright-ability. However, to avoid any question, a project may adopt a policy of requiring a signed CLA for every contribution, no matter how insignificant.

A common business case for use of a CLA in order to use different outbound licences is for projects run by a company where the code is offered under both an Open Source (usually a copyleft licence) and proprietary licence, often referred to as 'dual licensing'. In this case, using a CLA also allows the option of fully closing the code and ceasing to offer later versions under an Open Source licence.

Some Open Source projects may want to retain the ability to provide code under different Open Source licences or contribute the same code to other Open Source projects. For example, if the project code was to be used under either Apache-2 .0 or GPL-2.0 (due to these licences being deemed incompatible by the FSF), contributions under a CLA could allow such disjunctive outbound licensing. Of course, this result could also be achieved via inbound = outbound by accepting contributions under both licences.

As noted earlier, this power to differ the outbound licence comes at the cost of egalitarian nature of inbound = outbound projects, an important constituent of the collaborative model for many.

[10] <http://www.harmonyagreements.org/index.html> accessed 13 April 2022; <https://en.wikipedia.org/wiki/Project_Harmony_(FOSS_group)> accessed 13 April 2022.

[11] <http://contributoragreements.org/> accessed 13 April 2022.

[12] <https://fsfe.org/news/2017/news-20171013-01.en.html> accessed 13 April 2022.

Some developers simply refuse to contribute to CLA based projects for this reason; others may not bother with the hassle of signing something. There is no doubt that the use of a CLA carries an administrative cost. Where an entity is contributing by way of its employee representatives, tracking who is a current employee and authorised to contribute under the entity's CLA also needs to be taken into account and this also requires ensuring that CLAs are in the correct contributor's name.

4.2.4 Copyright assignment

A copyright assignment is not a licence. It is the full transfer of all transferable rights in copyright in the code such that the contributor relinquishes copyright ownership. Because full copyright ownership is transferred from the contributor to the project owner, the contributor can no longer exercise any of the rights associated with copyright ownership, such as copying, creating derivative works, further assignment, enforcing their copyright, etc. Rights like moral rights (the right to be identified as author of the code) which cannot be waived or assigned in some countries may not transfer.

Copyright assignment agreements often include a broad licence grant-back to the contributor, giving the contributors all rights other than ownership thus allowing the contributor to otherwise use the code. In many cases they may also licence the code as they choose. The contributor essentially becomes a licensee of the work they originally created and then contributed to the project, instead of its owner.

These agreements may also include a specific patent grant that follows the copyright in the contributions. Because copyright ownership is transferred, this also means that the right to enforce the licence lies with the project owner to which the copyright was assigned.

Assignment agreements may contain other contractual terms such as representations by the contributor or a promise by the project owner to use a certain outbound licence and so require scrutiny, particularly if not industry standard.

A full assignment can only be made from the originator of the code once. That is, if a contributor assigns code to one project, they cannot later assign it to another project, thus potentially limiting the ability for the same code to be contributed to multiple projects. While this possibility is remote due to the rare use of assignment agreements, it is perhaps an unanticipated hindrance that should still be considered.

Similar to a CLA, there needs to be a legal entity with which to execute the assignment agreement and to which the intellectual property (IP) rights are transferred. Again, this could be an individual person, a non-profit entity, or a corporation. In most jurisdictions, assignments must be executed in writing and in some have the added complexity of requiring to be 'delivered as a deed'.

Perhaps the most well-known use of an assignment agreement is for the GNU projects maintained by the FSF. Despite being a non-profit charity with a mission for supporting free software, the assignment agreement has caused much consternation among developers over the years.[13] FSF's stated rationale for using rests on FSF's ability to register the copyright work in the US more easily and to enforce the licence.[14] Given the reception to the FSF's assignment agreement over the years, most other projects have steered away from this option.

4.2.4.1 Why use it?

An assignment agreement provides ultimate control for the project owner. Assuming the agreement does not place obligations on the project owner, the project owner can use any outbound licence, register the copyright in the US, and enforce the copyright in the contributions as if it wrote the code itself.

Assignment agreements require the highest level of legal review and the highest burden on contributors. To an even greater extent than CLAs, there is limited standardised body of agreements or accepted text, although Project Harmony did create one.

Copyright assignments from each contributor to the project creates the greatest asymmetry in terms of rights. If developers are uncomfortable with CLAs, assignments are cause for even more heartburn.

Some companies or projects that used assignment agreements in the past have dropped them over time. Due to the friction assignments cause in a collaborative environment, Open Source counsel with experience will generally advise against their use but for the rarest of cases.[15]

4.3 Employee Contributions

Some lawyers like CLAs because they assume that an employee contributing on behalf of their employer will check with their legal department and get proper approval before agreeing to the CLA and contributing to the project. The rationale here is that employee contributors may not be authorised signatories for their employer and only authorised signatories can bind a legal entity. Thus, using a legal agreement (in the form of a CLA or assignment) that requires a signature will trigger legal review and the signature process within the contributing corporation. This then ensures that the proper authority was obtained for whatever IP rights were granted to the Open Source project via the contributor agreement.

[13] See <https://lwn.net/Articles/414523/> and <https://lwn.net/Articles/529522/> as examples, both accessed 13 April 2022.

[14] <https://www.fsf.org/bulletin/2014/spring/copyright-assignment-at-the-fsf> accessed 13 April 2022.

[15] For this reason, the next section mentions only CLAs.

From the perspective of the project owner, the concern here is that an employee contributing without the proper permission of their employer could result in that employer later trying to assert that it did not grant the rights associated with the licence for the Open Source project, potentially via an infringement suit.

For these reasons, a CLA (or assignment agreement) gives comfort over inbound = outbound where no formal signature is required for inbound contributions. Proponents of using CLAs for the purpose of obtaining proper authority for the reasons stated earlier put themselves in a bind; this view, taken completely, would mean only contributing to and consuming Open Source software that also uses a CLA. Such a position is untenable in today's software reality.

The rise of automation for signing CLAs confuses the goal of strict authority as it places the signature in the hands of the contributor by including in the CLA a representation that the employee/contributor has authority to sign, authorisation to bind their employer legally, or both as required by the nature of the relationship.

Such clauses put the onus back on the contributor to contribute only what they have the rights to. This is essentially the same as using the DCO. Similarly, if there is a later issue, the project has recourse against the employee/contributor, not the employer.

While this position regarding authority to bind an entity may hold technical legal merit, it ignores history and the practical reality of Open Source project governance and community norms.

In thirty years of the vast majority of Open Source projects using inbound = outbound, there have been precious few instances of an employer objecting after the fact to contributions made by employees resulting in a challenge for the project. Obtaining proper permission and an authorised signature assumes employers have a process to follow and employees follow it.

In reality, this may not be the case and employees may sign legal agreements, perhaps unwittingly, in order to make contributions and move on. While those employees may not have formal signing authority, they probably have enough apparent authority to create a binding agreement.

The proliferation of the inbound = outbound model has been a huge part of the success of Open Source software in allowing a low-friction path to collaboration. To call that into doubt based on an academic legal analysis ignores the broader picture and defies industry practice since the inception of Open Source software. Using a legal process (i.e. the process for obtaining an authorised signature) as the gatekeeper for Open Source contributions is neither practical nor good practice; lawyers need to work hard to train their developers and engineers about Open Source best practices, not assume that some rigid internal legal approval process will save them.

As corporate Open Source involvement has increased, it has become more common for companies to put in place appropriate internal processes for approval to contribute to Open Source projects via programs to manage, strategize, and promote their Open Source involvement.[16] Everyone can agree that this is the right approach and it is considered further at Chapter 21.

4.4 Practical Advice

4.4.1 Best approach?

The best approach for the legal agreement governing your Open Source project ultimately depends on the goals of the project. For the vast majority of projects, inbound = outbound works just fine. Any inbound agreement that introduces friction to the collaborative project should be chosen for a solid reason only and implemented in such a way that is consistent with the intended goal of its use.

When weighing the pros and cons of each approach for a given project, considerations beyond the legal technicalities must be included. Community considerations must weigh in, such as how the legal arrangement will be viewed by potential contributors to the project, common practices for the given community, long-term uses for the code, and so forth. Successful Open Source projects are not merely defined as code that lots of people use but as communities set up to foster collaboration. One of the most common entry points to an Open Source project community occurs when someone wants to contribute code. Thus, how contributions are handled is a first impression with a lasting impact.

Ideally, whatever the licensing model for a given Open Source project, there should be clarity as to what and why it is such, and be implemented in a way as to realise its goals.

When the licensing of an Open Source project is unclear, incomplete, or missing in any way, it wastes time and can ultimately hinder or prevent use or contributions. Clear and upfront information as to the licence(s) that apply(ies) to the project not only gives newcomers easy access to information, but also signals that this is an Open Source project.

In particular, using machine-readable standards for communicating this information (where possible) helps downstream users leverage automation of software management in Open Source licence compliance.

As discussed in more detail in subsequent chapters, advancements continue to be made in terms of providing outbound licence information or identifying such information with automated tooling, usually related to the goal of Open Source licence compliance. However, progress on getting projects to identify the inbound

[16] See, for example, programs to facilitate such activities: <https://www.openchainproject.org/> and <https://todogroup.org/> both accessed 13 April 2022.

licence more clearly for their project lags behind. Following are practical tips for communicating the licence information for Open Source projects, both inbound and outbound.

(1) **Identify the outbound licence**

  (a) Licence file: Place the complete text of the licence in its own file at the top-level directory or an appropriately named subdirectory if there is more than one licence text.

  (b) File-level licence notice: Place the SPDX-License-Identifier tag for the outbound licence in every file at or near the top of the file in a comment. The SPDX-License-Identifier syntax may consist of a single SPDX-Licence-Identifier or an SPDX Licence Expression to represent a single licence or a compound set of licences (respectively) that apply to that file. For more information on the use of SPDX identifiers, see <https://spdx.org/ids> or <https://spdx.org/ids-how> accessed 13 April 2022.

  1.3. Identify the outbound licence in your README and project website (if applicable). Include a concise statement as to the outbound licence in your README, preferably in a section called 'licence' and link to your LICENCE file. The same statement can be used on your project website, if you have one.

2. **Identify the inbound licence**

  (a) Include a copy of the inbound licence. If your project uses a different licence or agreement than the outbound licence, such as a CLA or assignment agreement, include a full copy of the text of that agreement at the top-level directory.

  (b) Identify the inbound licence in your README and project website (if applicable). Include a concise statement as to the inbound licence in your README, preferably in a section called 'licence' and link to the relevant file. The same statement can be used on your project website, if you have one.

  (c) Contributing file. Place a file at the top-level directory called CONTRIBUTING, and include the inbound licence information with a link to the agreement. This file can also include information about how to contribute to your project, coding standards, etc.

Examples of licence-related README statements, which would also include links in the appropriate places:

> "This software is provided under the BSD 3-Clause licence. Contributions to the project are accepted under the same licence."

"This software is provided under the BSD 3-Clause licence. Contributions to this project are accepted under the same licence with developer sign-off under the Developer's Certificate of Origin as described in Contributing."

"This software is provided under the BSD 3-Clause licence. Before you contribute, you will need to sign the Contributor Licence Agreement."

5
Copyright Enforcement

Miriam Belhausen

5.1 Introduction

The enforcement of rights in Open Source began with the odd case being brought, but these enforcement efforts have constantly increased since the mid-2000s.[1] This is especially true for Germany, where for the last decade there has been a high double-digit number of cases per year.[2] An outline of the cases and court decisions, which enable strategic and disruptive enforcement around Open Source licensing, the protected usage of code, and the copyright in that code is set out below (for

[1] An overview over the existing case law globally is available in Heather Meeker, *Open Source for Business*, 2nd edn (Kindle Direct Publishing Platform, 2017) Chapter 19; an overview especially over German case law is available from the '*Institut für Rechtsfragen der Freien und Open Source Software*' (in English: Institute for Legal Issues of Open Source) under <<https://ifross.github.io/ifrOSS/Cases. Additionally, there were further cases, such as *Wallace v Free Software Foundation*, where the US District Court for the Southern District of Indiana dismissed a claim based on antitrust violations of the FSF; see <https://cyberlaw.stanford.edu/packets003771.shtml> accessed 14 April 2022, or several cases against several companies based on copyright in BusyBox, or by Artifex Software, Inc. based on copyright in MuPDF. However, the main purpose of the chapter is to highlight decisions and aspects which are not only relevant under specific circumstances but lay the groundwork for enforcing copyright in Open Source.

[2] This is a conservative estimate based on available file numbers, however, most of these cases are settled out of court.

Miriam Ballhausen, *Copyright Enforcement* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Miriam Ballhausen 2022. DOI: 10.1093/oso/9780198862345.003.0005

summaries of other Open Source-related court decisions, please see Chapter 3, especially under sections 3.1 and 3.4.1).

Most of these cases follow the same pattern. They are initiated through a cease and desist letter sent by the copyright owner to a company (cases against individuals are less common) using their Open Source-licensed code and who is perceived to be infringing that code's Open Source licence, requesting that the licensee refrain from using the specific Open Source code unless the applicable licence is complied with. The cease and desist letter will also set a short deadline for the licensee to sign a declaration that they will cease and desist from the perceived infringing behaviour and notifies the licensee/user of the copyright holder's intent to pursue their claims against the infringing behaviour in court, unless the declaration is signed.

With that in mind, Open Source-related copyright enforcement (like many areas of copyright enforcement) has become highly standardised to a point where often largely identical cease and desist letters are sent to various recipients.[3] This has an impact of commoditising the process and reducing costs, making these enforcement actions much more accessible to individual developers and community-driven projects.

Against this background, this chapter will focus on the 'What?', 'How?', 'Why?', and 'Who?', as well as the key arguments in relation to the enforcement of copyright in Open Source:

(1) What an enforceable copyright infringement is.
(2) How can copyright in Open Source be enforced?
(3) Why are copyright in Open Source so consistently enforced in Germany?
(4) Who can enforce copyright in Open Source?
(5) What are the key arguments, alleged infringements, and court decisions on licence compliance?

Each of these questions can be answered and discussed extensively for every legal system and especially with regard to specific procedural aspects of German law. Respective overviews and discussions are regularly available in law journals and law reports.[4] This chapter is intended to answer these questions more broadly and especially to provide an overview over what to consider when facing or considering bringing an Open Source-related copyright enforcement action.

[3] This occasionally even includes copy-pasting the allegations and the file number.

[4] See, e.g., Marcus von Welser, 'Opposing the Monetization of Linux: McHardy v. Geniatech & Addressing Copyright "Trolling" in Germany' (2018) 10(1) *Journal of Open Law, Technology & Society* 9–20, available at <https://jolts.world/index.php/jolts/article/view/128>, or Till Jaeger, 'Praktische Umsetzung von Lizenzbedingungen der GNU General Public Licence (GPL) und Grenzen ihrer Durchsetzbarkeit' (2019) Computer & Recht 765–9, which includes detailed discussions on the calculation of contractual penalties based on declarations to cease and desist under German law.

5.2 What is Copyright Infringement and What Claims Can Be Made?

For copyright in Open Source to be enforceable, the following aspects need to be considered. Software in general and Open Source in particular need to be protected by copyright law (see section 5.2.1) in favour of the copyright holder (see section 5.5). This copyright needs to be generally enforceable (see section 5.2.3), even though rights of use to the Open Source are broadly granted by the copyright owner under any Open Source licence (see section 5.2.2).

5.2.1 Copyright protection of software

Software—usually referred to as 'computer programs'[5] in the respective legal texts—is widely protected as literary work (i.e. the same as books or this chapter for example) under applicable copyright law, provided that the software embodies an author's original creation.

Such copyright protection has been assumed for application programing interfaces (APIs) by the US Court of Appeals for the Federal Circuit in 2014. This decision is part of recently resolved legal dispute between Oracle America, Inc. (Oracle) and Google, Inc. (Google), in which Oracle claimed copyright and patent protection for several APIs, which are part of the Java technology. The technology had originally been owned by Sun Microsystems, which was purchased by Oracle. The APIs were included in earlier versions of the Android operating system. While the copyrightability of APIs was (re-)considered by the US Supreme Court, the Court was considered whether Google's use of the APIs constitutes fair use, which had been argued by Google, but rejected by the lower instance court. The detail of are discussed in Chapter 3.

All rights of use to the software (e.g. the right to distribute, the right to modification) initially lie with the software developer.[6] Third parties (i.e. anyone who is not the software developer) may only use the software, if and to the extent rights of use are granted (licensed) to them.

5.2.2 Open Source licensing

Rights of use to software are granted through licences. Such rights may be granted in various ways, including as simple/single right of use with others having the same

[5] In line with the Model Provisions made available by the World Intellectual Property Organization (WIPO) a computer program is 'a set of instructions capable, when incorporated in a machine-readable medium, of causing a machine having information-processing capabilities to indicate, perform or achieve a particular result'; Model Provisions available at http://www.wipo.int/mdocsarchives/AGCP_NGO_IV_77/AGCP_NGO_IV_8_E.pdf> accessed 14 April 2022.

[6] Exceptions to this rule may apply depending on the applicable copyright law (e.g. in case of employment relationships the rights of use may lie with the employer).

rights, or exclusively so that the licensee is the only one who may lawfully exercise a particular right or as a sole licence where the licensee is granted a simple/single licence, but is in a similar position to the exclusive licensee, because apart from the licensee only the licensor may continue to use the software.

A licence may be territorially restricted (e.g. for the European Union (EU) only) or granted worldwide; it may be restricted in time (e.g. for a year, as happens with hardcover books, before they become available as paperback) or it may be perpetually granted, it may be RF or paid, and may be incumbered by a field of use restriction or granted for varying types of use, for example for the reproduction in whole or in part, for the translation, adaptation, arrangement, or other modification, for the distribution of the computer program or for making it publicly available, or licences may be granted to cover all of these types of use.

These licensing options always exist irrespective of whether software is licensed on a proprietary basis or as Open Source and 'but all OSI-approved licenses are perpetual'.

All Open Source licences make use of these options in a particular way so that rights of use are granted to the farthest extent possible to ensure that every licensee has, as described in the Free Software Foundation's (FSF) Four Freedoms:

(1) The freedom to run the program as desired, for any purpose.
(2) The freedom to study how the program works, and change it so it does computing as desired.
(3) The freedom to redistribute copies.
(4) The freedom to distribute copies of modified versions, thus giving the whole community a chance to benefit from changes made.[7]

and in the OSD as explained in Chapter 3.

5.3 Enforceability of Open Source Licences and Termination Provisions—How?

At the outset of the enforcement of copyright in Open Source, the broad grant of rights and freedoms in the licences was regularly used to argue that the copyright holder waived all their rights of use, which initially exist (see section 5.2.1), when licensing software under an Open Source licence. This argument was put to a test by Sitecom, who had been sued by Harald Welte for the infringement of his

[7] These criteria, referred to as the four freedoms, were defined by the Free Software Foundation to determine, if licence terms qualify as free; see 'What is free software', available at <https://www.gnu.org/philosophy/free-sw.en.html> accessed 14 April 2022. They are similar to the ten criteria that were later identified by the OSD to qualify software as being licensed as Open Source software; see 'The Open Source Definition', available at <https://opensource.org/osd> accessed 14 April 2022.

copyright in netfilter/iptables, which is part of the Linux kernel. The court ruled in favour of the enforceability of Open Source licences and held that:

> one cannot perceive the conditions of the GPL … as containing a waiver of copyright and related legal rights. To the contrary, the users … rely on the concept of copyright and the copyright law in order to protect and secure their understanding of how software must be developed and distributed going forward.[8]

This argument was confirmed and applied also by Welte./.Versatel, where the Regional Court of Berlin also clarified that licensing software under the GPLv2.0 does not mean that rights to this software are waived.[9]

In Welte./.Sitecom, the Regional Court in Munich further clarified that the enforceability of Open Source licences and especially the obligations they define, is not excluded by the (especially strict) German laws on general terms and conditions.[10] These laws define both formal[11] and content requirements, especially prohibiting specific terms.[12] In addition to these specific provisions, German law generally prohibits all provisions, which are 'unreasonably disadvantageous' for the party, to whom the general terms and conditions are proposed.[13] In case of Open Source licences, this is the licensee.

In light of these requirements, all Open Source licences include terms, which are invalid under German law and would therefore generally not be enforceable. However, in Welte./.Sitecom, the court concluded that this was not the case for the termination of rights in (e.g. in Section 4 GPLv2.0). Despite Sitecom's argument, the court held that the termination was not unreasonably disadvantageous for the licensee and was therefore valid and enforceable. As it only applied in cases where the licensee did not comply with the licence's requirements, the termination's negative effects did not unreasonably disadvantage the licensee. In any event, the court held, the licensee could not 'cherry pick', claiming that the Open Source licence was validly granting rights of use but invalid with regard to the obligations it put forward and the conditions under which the rights of use were granted. Given the tight connection between the grant of rights and the requirements, the court held,

[8] District Court of Munich, decision dating from 19 May 2004, file number 21 O 6123/04—Sitecom./.Welte, available at <https://www.ifross.org/Fremdartikel/judgment_dc_munich_gpl.pdf > (in English) and <https://www.ifross.org/Fremdartikel/urteil_lg_muenchen_gpl.pdf> (in German), both accessed 14 April 2022.

[9] Regional Court of Berlin, decision dating from 21 February 2006, file number 16 O 134/06—Welte./.Versatel, available at <https://www.telemedicus.info/urteile/Urheberrecht/Open-Source/556-LG-Berlin-Az-16-O-13406-Verstoss-gegen-GPL-WLAN-Router.html> (in German).

[10] The relevant provisions are included in sections 305 et seq. of the German Civil Code. An English translation is available here: <https://www.gesetze-im-internet.de/englisch_bgb/>.

[11] Section 305 German Civil Code, for example, defines how terms and conditions need to be made available to even become part of a contractual agreement.

[12] Section 309 German Civil Code, for example, excludes limitations of liability to a large extent.

[13] The respective provision can be found in Section 307 German Civil Code.

any argument in favour of the licence obligations being invalid would need to be applied to the Open Source licence as a whole and would include the grant of rights as well, thus leaving the defendant entirely without a licence and therefore even more certainly in breach of copyright.

5.4 Why is Copyright in Open Source so Consistently Enforced in Germany?

As is clear from Open Source cases across the globe, copyright in Open Source can be enforced under many jurisdictions worldwide. Nonetheless and as mentioned on the outset of this chapter, Germany sees a disproportionately high number of enforcement cases which regularly involve organisations that are non-compliant in other jurisdictions, not only in Germany.

This begs the question of why copyright in Open Source is so consistently enforced in Germany.

The main reason for this territorially focused enforcement lies in the procedural laws which are favourable to claimants for several reasons. Most importantly, copyright can be enforced in preliminary proceedings, serving the purpose of protecting the right holder[14] in cases where an infringement of the rights is either imminent or already happening. In these cases, the right holder can file for a cease and desist order once he has requested the infringer to cease and desist from (further) infringements by means of a cease and desist letter. This cease and desist letter needs to demand that the infringer signs a declaration to cease and desist with penalty provision. If the infringer does not comply with the right holder's demand to sign a declaration to cease and desist, such an order can be filed for. The cease and desist order may then be passed within days and often without the infringer being involved. To ensure a fair hearing, the infringer may later object to the court order, however until a date for an oral hearing is set, the infringer needs to abide by the court order and will normally not be able to continue to distribute the affected products.

As these orders are passed in such speedy proceedings and given the lowered burden of proof for the claimant, cease and desist orders are only valid for six months, after which they expire. This time period, or rather the risk of being ordered to cease and desist from the distribution of products for such a period of time and its impact on the user's products and potential impact on the supply chain, often suffices to put enough pressure on the defendant to react to any copyright owner's claim to cease and desist from an infringement of Open Source licences in Germany.

[14] These options are not only available to copyright owners but for all other right holders as well, provided that there is a need for preliminary and immediate protection to avoid further infringements.

Where the copyright owner's key focus is on ensuring that an organisation complies with Open Source licence terms, such preliminary proceedings are an effective tool. However, where proceedings are used to obtain as much money as possible out of a court settlement or declaration to cease and desist as possible, thus allowing the copyright owner to trigger penalty payments at will, there is a high risk that the enforcement may become formalistic and that a copyright owner may suggest very detailed licence interpretation, which may both be impractical and harmful for the Open Source community at large. To safeguard against this, the Software Freedom Conservancy,[15] the FSF,[16] and the Netfilter project,[17] organisations which hold code on behalf of developers, whose work is detailed further in Chapter 18, and whose code is often the subject of enforcement actions relating to Open Source, defined principles of community-oriented General Public License (GPL) enforcement, aiming to ensure that any enforcement action taking by individual Open Source copyright owners is consistent in ensuring compliance and that it is not centred on generating payments.

5.5 Who Can Enforce Copyright in Open Source?

The third key question relates to who can drive the enforcement. The answer to this question has implications from a material and a procedural law perspective.

5.5.1 Ownership of copyright

As a general rule, the copyright is owned by the author of the code, that is the individual developer, unless:

(1) The code was developed as work for hire, so that the employer is considered the author even if an employee actually created the work;
(2) the copyright was assigned; or
(3) the (commercial) copyright was (exclusively) attributed or licensed to a third party.

Whether any, some, or all of these exceptions are relevant depends on the respective jurisdiction and the applicable copyright law.

[15] See 'The Principles of Community-Oriented GPL Enforcement', available at <https://sfconservancy.org/copyleft-compliance/principles.html> accessed 14 April 2022.

[16] See 'The Principles of Community-Oriented GPL Enforcement', available at <https://www.fsf.org/licensing/enforcement-principles> accessed 14 April 2022.

[17] See 'The statement of netfilter project on GPL enforcement', available at <https://www.netfilter.org/files/statement.pdf> accessed 14 April 2022.

With regard to Open Source projects, all of the concepts, which are roughly outlined earlier, may be relevant. There may be an individual developer licensing an entire piece of code under an Open Source licence. Code may also have been developed by employees of a company and/or developers engaged to create code as a work for hire. Subsequently, the code may have been licensed as Open Source by the employing or hiring entity, respectively the assignee. However, Open Source projects generally follow a collaborative approach and receive contributions from various developers. The set-up of this collaboration has a bearing on the copyright ownership and, more precisely, on who may drive enforcement around copyright.

Very roughly outlined, Open Source projects fall into the following three categories, where the following are likely to apply:

(1) If separate pieces of code or individual programs are brought together in one project, the developers of each separate piece of code or each individual program are likely to be individual and separate owners of copyright in their compound work.

(2) If code is developed jointly by several developers, these developers may jointly hold the copyright to the developed code, provided they intend to create a joint program, defined a common task to which each of them contributes, their development followed a joint idea, and cannot be commercialised individually across the individual contributions.

(3) Thirdly, contributions within Open Source projects may build on, expand, and amend the pre-existing code. In this case, both the pre-existing code, the contribution (provided it surpasses the threshold for copyright protection), and the work formed by the pre-existing code and the contribution are individually protected by copyright. For the third category the jurisdiction and applicable law are likely to have a bearing on the copyright ownership in the work created from the pre-existing code and the contribution(s). Depending on the jurisdiction and applicable law, the work created from pre-existing code and the contribution(s) may, for example, qualify as adaptation (e.g. under Section 9 German Copyright Act) or as collective work (e.g. under the US Copyright Act 17 USC § 101). The qualification as adaptation or collective work in turn determines the copyright ownership in the work and ultimately in the Open Source project as a whole. In case of an adaptation, the copyright in the adaptation are owned by the creator of the adaptation, who is of course dependent on a licence covering the adaptation from the owners of the copyright in the pre-existing works. If the work created from the pre-existing code and the contribution(s) qualifies as collective work, as is assumed for the Linux kernel under US law,[18] the

[18] See Linus Torvald's statement available at: <https://ipfs.io/ipfs/QmdA5WkDNALetBn4iFeSepHjdLGJdxPBwZyY47ir1bZGAK/comp/linux/collective_work_copyright.html> accessed 14 April 2022.

copyright in the collective work are owned by the collector, that is the creator of the collective work, who is arguably the initiator of the Open Source project.

Many Open Source projects, especially those which are sufficiently complex, will include a mix of the three types of collaborations and thus copyright ownership,[19] but the differentiation between them must be kept in mind as it impacts who may drive the enforcement. Although there are some exceptions under some copyright laws, as a general rule the enforcement may be driven by the copyright owner only. Accordingly, in case of individual ownership of copyright, the individual owner may drive the enforcement. In case of joint ownership, though, enforcement may require a joint action from all copyright owners or at least demand claims to be filed so that performance is due to all joint copyright owners. In the third scenario, the enforcement may generally be driven by each individual developer for their individual contribution and provided it surpasses the creativity threshold. This is the case with the enforcement actions in Germany, although courts have occasionally rejected claims, because the copyrightability of the individual contribution had not been sufficiently demonstrated.[20] Furthermore, depending on the type of collaboration and the set-up of the project, the author of the code may transfer or assign their rights to the project. In case of such transfer and assignment, the copyright enforcement may only be driven by the project as such or, depending on the setup at hand, by the recipient of the rights.

5.5.2 Enforcement of copyright

From a procedural perspective, the key questions particularly relate to the copyright owner's standing in court. To have standing, meaning the ability to enforce copyright in court, the claimant needs to be able to prove ownership of copyright in the Open Source (project) whose licence they claim is infringed.

In Hellwig./.VMWare, the Higher Regional Court in Hamburg laid out what this requires with regard to Open Source, demanding the claimant to prove the following:

(1) The claimant needs to *prove that he holds rights to the Open Source*, for which the licence has been (supposedly) infringed. In case of contributions to an Open Source project, this requires the claimant to demonstrate precisely

[19] Till Jaeger and Axel Metzger, *Open Source Software*, 5th edn (Nördlingen: C.H. Beck) para 203.

[20] Higher Regional Court Hamburg, decision dating from 28 February 2019, file number 5 U 146/16—Hellwig./.VMWare.

which contributions he made and which parts of the entire Open Source project's code these contributions relate to.

(2) Secondly, the claimant needs to show that these *contributions (alone) are a copyrightable work* under applicable copyright law. This usually requires information on and proof of the specific functionality of the code within the Open Source project, whereas proving that the Open Source project as such surpasses the threshold was found to be insufficient. An argument to the contrary, the court held, could not be based on the German Federal Court of Justice's '*Fash 2000*' decision, according to which 'complex' software was assumed to meet the requirements of copyrightable software under the German copyright act, unless proven otherwise. According to the Higher Regional Court of Hamburg, this assumption applies only in cases where the contribution itself constitutes complex software. Where that is not the case, the full burden of proving the contribution as such is copyrightable lies with the claimant.

(3) Thirdly, to have standing, the claimant needs to *prove that his contributions, which surpass the threshold of copyrightable work, were used by the defendant in their products*.[21]

In Hellwig./.VMWare, the Higher Regional Court in Hamburg held that this requires the claimant to show specifically which of his contributions to the original Open Source project the defendant is using in their products. Merely providing the entire code the defendant is using and information on contributions to the original Open Source project does not suffice, as it cannot be inferred from them that parts of the claimant's code which are separately protectable under (copyright) law were used by the defendant. The same, the court held, applies regarding header files in which the claimant is named as the header files alone do not prove that the defendant uses contributions made by the claimant or that these contributions surpass the threshold of copyrightable works.[22]

In its decision dating from 20 November 2017, the Regional Court in Hamburg[23] found the claimant's argument that extracted strings were matched to verify that a component was included in the defendant's products to be sufficient.[24] The defendant, on the other hand, was required to clarify and prove which components

[21] Higher Regional Court Hamburg, decision dating from 28 February 2019, file number 5 U 146/16—Hellwig./.VMWare.

[22] Higher Regional Court Hamburg, decision dating from 28 February 2019, file number 5 U 146/16—Hellwig./.VMWare.

[23] Regional Court of Hamburg, decision dating from 20 November 2017, file number 308 O 343/15.

[24] The claimant had additionally argued that copyrightable contributions of his were mandatory for the product to be network-compatible and therefore had to be included in the defendant's products. The appeal against the Regional Court of Hamburg's decision is currently lies with the Higher Regional Court of Hamburg, file number 5 U 231/17.

were included in the product[25] if he wanted to argue that the claimant's component were not included. Merely challenging that the components to which the claimant held copyright were included in the defendant's product was not sufficient, the court held. Given that the relevant software was a key component of the products the defendant was selling, the court argued that he would have been easily able to clarify which components it consisted of and thus to demonstrate if, contrary to the claimant's position, the claimant's component was in fact not included. The argument that the defendant would have had to engage external expertise to make such determination[26] was dismissed.

5.6 What Are the Key Arguments and Alleged Infringements?

As Germany has seen so many cases revolving around the enforcement of copyright in Open Source, German courts have ruled on several aspects of many key licence requirements, especially on the interpretation of (i) the obligation to provide the complete corresponding source code and (ii) the obligation to accompany the product with the licence text.

At the outset, the courts' position had always been that the burden of proof for the infringement lies with the claimant. In a more recent case, the Higher Regional Court in Hamburg held, though, that the defendant must prove his right to use the Open Source[27] and thus to present facts and evidence of his compliance with the respective licences' obligations.

5.6.1 Obligation to provide the source code

5.6.1.1 Complete corresponding source code

Under all Open Source licences, a key factor in compliance with the obligation to provide the source code is for the source code to be '*complete corresponding*'. This requirement was first interpreted by the Regional Court in Hamburg in 2013,[28] which ruled that for the source code to be complete corresponding, the following must be met:

- The compilation date of the firmware in the product may not be earlier than the date in the source code that is provided along with the firmware;

[25] Regional Court of Hamburg, decision dating from 20 November 2017, file number 308 O 343/15.

[26] In the case at hand the defendant was only selling products he had imported.

[27] Regional Court of Hamburg, decision dating from 20 November 2017, file number 308 O 343/15.

[28] Regional Court of Hamburg, decision dating from 14 June 2013, file number 308 O 10/13.

- The version numbers of the components in the firmware in the product need to match the version numbers of the respective components in the source code.

In 2017, the Regional Court in Hamburg further clarified that the scripts to control compilation and installation (as defined by the GPLv2.0) need to be included for the source code to be complete corresponding.[29]

5.6.1.2 Written offer

In the latter case, the court in Hamburg then further ruled on the requirements a written offer needs to meet.[30]

The claimant had used the following written offer, which the court found to be incompliant with the GPLv2.0's requirements.

> This product contains Free Software which is licensed under the GNU General Public License. After you purchase this product, you may procure, modify or distribute the source code of the GPL/LGPL software that is used in this product.
>
> If you contact our Support Center, we will provide you with a CD-ROM of the source code that is used, charging only the actual expensed involved. However, please be Side that we cannot provide guarantee with the source code, and there is also no technical support for the source code from us.

The court based its finding mainly on the offer's restriction, that the source code could only be requested 'after you purchase this product'. This, the court ruled, created the impression that only customers could request a copy of the source code, while the GPLv2.0 required the offer to be made to any third party.

A clarification that the offer was valid for at least three years was not found to be necessary, though. The court argued that the GPLv2.0's wording requiring the licensee to 'accompany … [the binary code] with a written offer, valid for at least three years', was ambiguous and thus needed to be interpreted to the licensor's (claimant's) disadvantage.[31] The provision, the court ruled, could be interpreted to require the licensor only to uphold the written offer for at least three years, while it did not necessarily require the licensor to specify the duration of validity in the written offer itself.

Finally, the court ruled that the written offer is only formally made if the text is immediately accessible, easily identifiable, and constantly available with the binaries.[32]

[29] Regional Court of Hamburg, decision dating from 20 November 2017, file number 308 O 343/15.

[30] Regional Court of Hamburg, decision dating from 20 November 2017, file number 308 O 343/15.

[31] The court arrived at this conclusion based on the German law on general terms and conditions, especially Section 305c para 2 BGB, according to which any ambiguous provision must be interpreted to the disadvantage of the party who is proposing the general terms and conditions.

[32] Regional Court of Hamburg, decision dating from 20 November 2017, file number 308 O 343/15.

5.6.1.3 Offering the source code for download

The court then applied this test to cases where the firmware was offered online for download. In this case, the Regional Court in Hamburg ruled, it would generally suffice if the written offer or the source code itself was available via link only, provided that the average user can easily determine that the source code is available via the link. For that, the actual download option cannot be further than two clicks away from the site where the firmware is made available, and the references must be clearly marked and placed in close proximity to the firmware. Making the source code available on a separate page, which is used to provide download options for the source code of various products, was held to be insufficient, unless there was a direct link from the firmware download to the complete corresponding source code.[33]

In any event, providing the source code for download only suffices to comply with the obligations if the firmware is also (only) offered for download. For firmware which is distributed as part of a product, the source code needs to be with the product. Referencing a download option in the product's manual, for example, does not suffice.[34]

5.6.2 Obligation to provide the licence text

The obligation to provide the licence text was first interpreted by German courts in Welte./.Skype, where the Regional Court in Munich held that merely referencing the GPLv2.0 did not suffice because it required the user to research the terms of the licence. Therefore a (full) copy of the licence text had to be made available with the product, respectively the firmware it contains.[35]

The licence text needs to be available in full. This, the court held in 2017, includes the preamble and the provisions under 'How to Apply These Terms to Your New Programs', because section 2 GPLv2.0 required the licensor to provide a 'copy of this Licence', not 'a copy of these Terms and Conditions'. As indicated by the subheading 'End of Terms and Conditions', the preamble, and the actual licence terms form the 'Terms and Conditions', which jointly with the instructions on application form the 'Licence'.[36]

The Regional Court in Hamburg further clarified the requirements, especially in cases where the firmware was offered for download. They held that either the full

[33] Regional Court of Hamburg, decision dating from 20 November 2017, file number 308 O 343/15.

[34] LG München I, Urteil vom 12.07.2007—7 O 5245/07.

[35] Regional Court of Munich I, decision dating from 12 July 2007, file number 7 O 5245/07—Welte./.Skype, available at: <https://www.telemedicus.info/urteil/lg-muenchen-lizenzverletzung-der-gpl/> (in German) accessed 21 July 2022.

[36] The court further concluded that the failure to include the instructions at the end of the licence also meant that the obligation to provide the disclaimer was not complied with.

licence text needs to be available directly on the website from where the firmware can be downloaded, or this full licence text needs to provide a direct link to it from the download page. The link was only found to suffice if the full licence text was immediately accessible, easily identifiable, and constantly available, meaning that the average user needs to be able to access the licence information via the link, which therefore needs to be in close proximity to where the firmware can be downloaded. To achieve sufficient clarity, the link needs to be clearly marked as a reference to the licence. However, it was found to be sufficient if the link referenced a manual, as users expect manuals to include the licence terms as well.[37]

5.7 New Trends

More recently, Open Source users and the Open Source community at large have started to challenge the claims they are repeatedly facing more aggressively. There are two key trends that have been developing over the last years. The GPL cooperation commitment and the OpenChain project.

5.7.1 Cure commitment

Under section 4 GPLv2.0 the licensor's rights are terminated with immediate effect in a case of incompliance. In contrast, section 8 paragraph 3 GPLv3.0 allows for a grace period of thirty days to come into compliance in case of a first-time notification of a licence violation.[38] The licence is not terminated, if the violation is resolved within thirty days following notification of non-compliance.

The GPL cooperation commitment aims to achieve the same for Open Source licensed under the terms of the GPLv2.0. It is closely connected to the Linux Kernel Enforcement Statement, by which the GPLv3.0's cure provisions were adopted for the Linux kernel by many but not all contributors.[39] They acknowledge the right of every contributor to enforce their rights individually but stated that they wanted to ensure that any such enforcement was conducted in the communities' best interest and therefore allowed for a grace period of thirty days to come into compliance. If observed, any enforcement actions are intended to be inadmissible.

[37] Regional Court of Hamburg, decision dating from 20 November 2017, file number 308 O 343/15.

[38] Section 8 para 3 GPLv3.0 reads: 'Moreover, your licence from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this Licence (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.'

[39] <https://www.kernel.org/doc/html/v4.16/process/kernel-enforcement-statement.html> accessed 14 April 2022.

In light of the Regional Court of Halle's decision on the GPLv3.0's cure provision,[40] neither the cure commitment nor the enforcement statement are likely to effectively exclude enforcement actions, even in case of a first-time violation that is cured within thirty days. The licensor has a reasonable interest in effective prevention of future infringements, which under German law requires a declaration to cease and desist including a penalty provision. Accordingly, even if the cure commitment and the enforcement statement are globally adopted, enforcement are likely to continue as before in Germany.

5.7.2 OpenChain

In contrast to the cure commitment and enforcement statement, the OpenChain project discussed in detail in Chapter 6 takes a more global approach, aiming to make compliance with applicable Open Source licences simpler and more consistent and thus establishing trust in the Open Source from which software solutions are built. As the OpenChain specification defines key requirements of a high-quality Open Source compliance program, mere conformance with this specification does not exclude incompliance of a single software deliverable and will therefore not prevent cease and desist claims. However, the ability to demonstrate compliance with the project's specification may be effectively used by the licensee to meet the burden of proof obligations as defined by the Regional Court in Hamburg (see introduction to section 5.5).

[40] Regional Court of Halle, decision dating from 27 July 2015, file number 4 O 133/15.

6
Transforming the Supply Chain with Openchain ISO 5230

Shane Coughlan

6.1 Overview

OpenChain ISO 5230 increases Open Source compliance in the supply chain. This issue, which may be incorrectly characterised as solely a legal concern or as low priority from a business perspective, is inherently tied to ensuring that Open Source is as useful as possible with as little friction as possible. In a nutshell, because Open Source is about the use of third-party code, compliance is the nexus of where equality of access, safety of use, and reduction of risk can be found. OpenChain ISO 5230 is built to increase trust between organisations to accomplish this.

Today many companies understand Open Source and act as major supporters of Open Source development. However, addressing Open Source licence compliance in a systematic, industry-wide manner has proven to be a somewhat elusive challenge. The global IT market has not yet seen a significant reduction in the number of Open Source compliance issues discoverable in areas like consumer electronics over the last decade.

Shane Coughlan, *Transforming the Supply Chain with Openchain ISO 5230* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Shane Coughlan 2022.
DOI: 10.1093/oso/9780198862345.003.0006

The majority of compliance issues originate in the midst of sharing multiple hardware and software components between numerous entities. The global supply chain is long, and the participants are simultaneously intertwined and disparate. It is perfectly possible to have companies making hardware, companies making software, and companies doing both collaborating around a relatively small component. The results in terms of products are often outstanding but the challenge of keeping track of everything is substantial.

6.2 Compliance is a Process Challenge that Spans Multiple Organisations

Open Source presents a specific challenge in the global supply chain. This is not because Open Source is inherently complex but rather due to the varying degrees of exposure and domain knowledge that companies possess. By way of example, a company developing a small component that requires a device driver may have staff entirely unfamiliar with Open Source. One mistake, one misunderstanding, and one component deployed in dozens of devices can present an issue. Most compliance challenges arise from mistakes. Few, if any, originate with intent.

Ultimately solving Open Source compliance challenges involves solving Open Source compliance in the supply chain. This is no small task: there are thousands of companies across dozens of national borders using numerous languages in play. The solution lies beyond the realm of inter-company negotiation. To address Open Source compliance challenges, the global supply chain must align behind certain shared approaches.

6.3 Because No Single Company Makes a Finished Device, No Single Company Can Solve Compliance Challenges

Awareness of this fact and the provision of a practical solution are two different matters. It takes time for ideas and suggested approaches to percolate and mature. It took input from lawyers and managers and developers and political scientists. It took, in short, a while for the ingenuity of the human community to bounce ideas back and forth until a simple, clear approach could be found.

6.4 The Best Solutions Are Often the Simplest, with the Lowest Barriers to Entry

The OpenChain Project formally launched in October 2016 and is hosted by the Linux Foundation. It originated in discussions that occurred three years earlier and continued at an increasing pace until a formal project was born. The basic idea was simple: identify key recommended processes for effective Open Source management. The goal was equally clear: reduce bottlenecks and risk when using third-party code to make Open Source licence compliance simple and consistent across the supply chain. The key intention was to pull things together in a manner that balanced comprehensiveness, broad applicability, and real-world usability.

6.5 OpenChain ISO 5230 is Intended to Make Open Source Licence Compliance More Predictable, Understandable, and Efficient for the Software Supply Chain

The OpenChain Project is building and disseminating an industry standard for licence compliance. It is designed to be the foundation for Open Source compliance in the supply chain. Engagement and adoption are simple, free, and supported by a vibrant community backed by leading multinationals across multiple sectors.

There are three interconnected parts to the OpenChain Project. A Specification that defines the core requirements of a quality compliance program. A Conformance method that helps organisations display adherence to these requirements. A Curriculum to provide basic Open Source processes and best practices.

6.6 A Simple Specification that Explains the Key Requirements of a Quality Compliance Program

The core of the OpenChain Project is the Specification. This identifies a series of processes designed to help organisations of any size address Open Source compliance issues effectively. The main goal of organisations using the OpenChain ISO 5230 Specification is to become conformant. This means that their organisation must meet the requirements of a certain version of the OpenChain ISO 5230 Specification. A conformant organisation can advertise this fact on its website and promotional material, helping to ensure that potential suppliers and customers understand and can trust its approach to Open Source compliance.

6.7 A Clear and Free Way to Check Conformance with the Specification

OpenChain ISO 5230 Conformance can be checked via a free online self-certification questionnaire provided by the OpenChain Project. This is the quickest, easiest, and most effective way to check and confirm adherence to the OpenChain ISO 5230 Specification. There is also a manual conformance document available for organisations whose process requires a paper review or disallows web-based submissions. Both the online and the manual conformance can be completed at a pace decided by the conforming organisation, and both methods remain private until a submission is completed.

6.8 Reference Material to Support Conformance and with Broader Questions of Training and Processes

OpenChain Project provides extensive reference material to help organisations meet certain aspects of the OpenChain ISO 5230 Specification. These provide a generic, refined, and clear example of an Open Source compliance process or support documents that can either be used directly or incorporated into existing company materials. OpenChain Project reference material is available with very few restrictions to ensure organisations can use it in as many ways as possible. To accomplish this, it is licensed as Creative Commons—Zero (CC-0), effectively public domain, so remixing or sharing the material freely for any purpose is possible.

6.9 Community and Support

The OpenChain Project provides what is believed to be a compelling approach to making Open Source compliance more consistent and more effective across multiple market segments. However, good ideas need implementation, and in the context of Open Source this inevitably hinges on the creation of a supporting community. The OpenChain Project at the time of writing has twenty Platinum Members that support its development and adoption, and a growing global community containing hundreds of companies.

At its core, the OpenChain Project is about providing a simple, clear method of building trust between organisations that rely on each other to share code and create products. Any organisation that is OpenChain ISO 5230 conformant is aligning behind key requirements that their peers agree are required in a quality compliance program. This is about confirming overarching processes and policies, while allowing the specifics of each process and policy to be crafted by each organisation to suit its specific needs.

6.10 Conclusion

The OpenChain ISO 5230 Specification is ready for adoption by any organisation that creates, uses or distributes free and Open Source code. The online conformance is free of charge, the mailing list and Work Team calls are open to everyone. Arguably, this is the first time a single, unifying approach to addressing the challenge of Open Source open compliance in the supply chain exists.

6.11 References

- OpenChain Project: <https://www.openchainproject.org/> accessed 14 April 2022.

7
SPDX and Software Bill of Materials ISO/IEC 5962L 2021

Kate Stewart

7.1 Why Create a Software Bill of Materials?

When the Software Package Data Exchange® (SPDX) project[1] was started in 2010, it was with a simple goal of being able to share summary information about a software package between the creator and consumer. At that time, to comply with the licences in Open Source, you had to find them in the source code. This resulted in hours of 'grep'ing' or working with commercial source scanning tools, and once you had the details, you didn't have a good way of sharing them. After comparing notes and recognising there was a group of managers, lawyers, and developers frustrated by the same problem, we started a grassroots effort to standardise the information that we wanted to share and it became hosted at the Linux Foundation. We needed to be able to capture the known information about Open Source software as well as proprietary software, as products are created from both.

Over the years, more use cases were identified that we wanted to share information about, and so additional capabilities were added to the SPDX specification.[2] We recognised early on that we wanted to be able to tell if the information we'd recorded about a package was stale, so we added the ability to record a cryptographic hash of the object being described. We wanted to know if the information was complete—had files been removed or added—so a verification code was included.

[1] <https://spdx.dev> accessed 22 June 2022.

[2] <https://spdx.github.io/spdx-spec/> accessed 22 June 2022.

Kate Stewart, *SPDX and Software Bill of Materials* In: *Open Source Law, Policy and Practice.* Edited by: Amanda Brock, Oxford University Press. © Kate Stewart 2022. DOI: 10.1093/oso/9780198862345.003.0007

We saw the need to record an arbitrary level of software components, so the definition of package was extended to encompass 'any group of elements' (which proved very useful for recording information about containers and all their layers). Being able to understand if there are vulnerabilities associated with a package is another use case that has been recognised in the last five years as an important reason to have available an accurate summary of software being used on a system. Once an accurate software bill of materials (SBOM) is made available with products, it simplifies the effort. An SPDX document is able to represent an SBOM[3].

7.2 What is an SPDX Document?

The SPDX specification defines a common language for communicating the components, relationships between components, licenses, security information, and copyrights associated with software. An SBOM needs to be precise and unambiguous in order accurately to identify the code being used in products and enable identification of any security vulnerabilities associated with that code. An accurate manifest contained in an SBOM also enables product creators to identify the license obligations. When we have a common language to communicate these concepts, information can be effectively shared, and it does not need to be regenerated at each step in the supply chain.

By providing a common syntax and vocabulary for organisations and communities to share this SBOM data, compliance can be automated, and this improved transparency facilitates vulnerability identification and remediation. Prior to the instigation of this standard SBOM within SPDX and its adoption industry, the need to meet an array of customer requirements to summarise the software metadata and licensing placed a huge barrier to entry and a burden on suppliers of Open Source software packages.

Before any concept is added to the SPDX specification it has to be added to the SPDX data model. Each SPDX bill of materials document is based on a full data model implementation and identifier syntax. This permits exchange between data output formats and formal validation of the correctness of the SPDX document. The project started off with two recognised file types, tag:value (.spdx) and Resource Description Framework in Attributes (RDFa). Over time, support to translate into spreadsheets was added. In the SPDX specification 2.2 release, the additional output file formats of JSON, YAML, and XML have been added to the formats supported in the 2.1 release (RDFa, tag:value, spreadsheet). Further information on the SPDX data model can be found in Annex C of the SPDX Specification, version 2.2,[4] and on the SPDX web site.[5]

[3] <https://ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom> accessed 22 June 2022.

[4] <https://spdx.github.io/spdx-spec/RDF-object-model-and-identifier-syntax/> accessed 22 June 2022.

[5] <https://spdx.org/rdf/terms/> accessed 22 June 2022.

Converting between multiple file types is made possible by having that underlying SPDX data model to guide the mappings.

7.2.1 Overview of an SPDX document

The SPDX specification describes the necessary sections and fields to produce a valid SPDX document. This grassroots effort has had participation over the years from a wide variety of software developers, systems and tool vendors, foundations, and the legal community, all committed to creating a common language for products, components, and software packages to be able to exchange SBOM data efficiently and effectively.

Each SPDX document can be composed from the following (see Figure 7.1):

- **Document Creation Information:** One instance is required for each SPDX document produced. It provides the necessary information for forward and backward compatibility for processing tools (version numbers, license for data, authors, etc.)
- **Package Information:** A package in an SPDX document can be used to describe a product, container, component, packaged upstream project sources, contents of a tarball, etc. It is just a way of grouping together items that share some common context. It is not necessary to have a package wrapping a set of files.

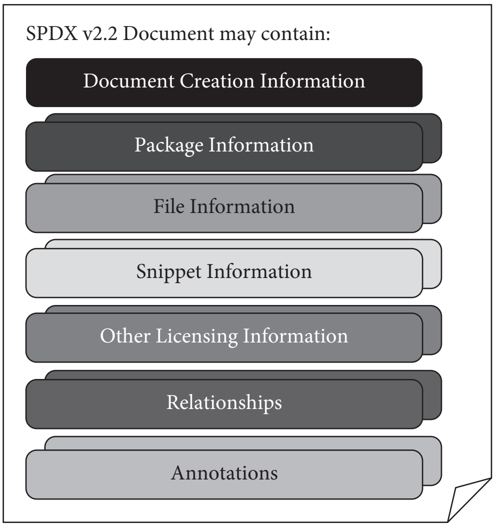
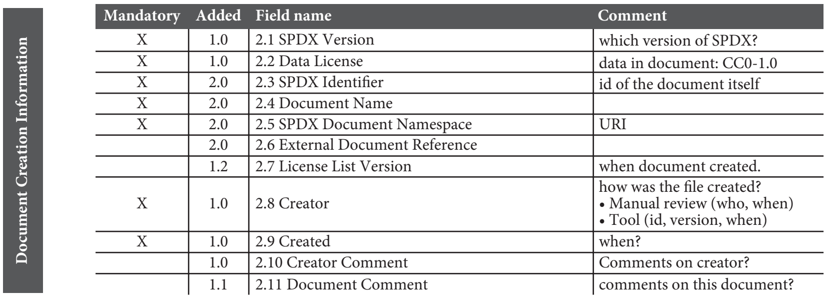

SPDX v2.2 Document may contain:
Document Creation Information
Package Information
File Information
Snippet Information
Other Licensing Information
Relationships
Annotations

Figure 7.1 SPDX Document Overview

- **File Information:** A file's important meta information, including its name, checksum licences, and copyright, is summarised here.
- **Snippet Information:** Snippets can optionally be used when a file is known to have some content that has been included from another original source. They are useful for denoting when part of a file may have been originally created under another licence.
- **Other Licensing Information:** The SPDX License List[6] does not represent all possible licences that can be found in files (such as uncommon or non-source-available licences), so this section provides a way to summarise other licences that may be present in software being described.
- **Relationships:** Most of the different ways that SPDX documents, packages, files, and snippets can be related to each other can be described with these relationships.
- **Annotations:** Annotations are usually created when someone reviews the SPDX document and wants to pass on information from their review. However, if the SPDX document author wants to store extra information that doesn't fit into the other categories, this mechanism can be used.

The only section that is mandatory in the SBOM, is the 'Document Creation Information' section for each document, all the rest are optional (see Figure 7.2). The creator can choose which sections (and subset of the fields in each section) that describe the software and metadata information to be shared.



Document Creation Information

Mandatory	Added	Field name	Comment
X	1.0	2.1 SPDX Version	which version of SPDX?
X	1.0	2.2 Data License	data in document: CC0-1.0
X	2.0	2.3 SPDX Identifier	id of the document itself
X	2.0	2.4 Document Name	
X	2.0	2.5 SPDX Document Namespace	URI
	2.0	2.6 External Document Reference	
	1.2	2.7 License List Version	when document created.
X	1.0	2.8 Creator	how was the file created? • Manual review (who, when) • Tool (id, version, when)
X	1.0	2.9 Created	when?
	1.0	2.10 Creator Comment	Comments on creator?
	1.1	2.11 Document Comment	comments on this document?

Figure 7.2 Document Creation Information

7.2.1.1 Document Creation Information

There must be a 'Document Creation Information' section for each SPDX document. In it, seven of the fields are required to be filled out. The version of the SPDX specification used to generate the document is the first field, as it provides the key

[6] <https://spdx.org/licenses/> accessed 14 April 2022.

to understand which fields are in each document. Each SPDX document is required to be under the CC0-1.0 licence,[7] and this is denoted by the ‘Data License’ field. Other mandatory elements in this section are the self-identification of the document and the ‘Document Namespace’, as well as who created the document and when.

Each field has a specific grammar associated with it and rules for parsing. Details of each field, rationale for the field, and parsing guidance can be found in the Document Creation Information section of the specification.[8]

An example of this section expressed as tag:value is:

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: SPDX document for Time version 1.7
DocumentNamespace:http://spdx.org/documents/d3e9fef0-00a0-4b39-
   bb28-ff3dc75c7200
LicenseListVersion: 2.5
Creator: Tool: Source Auditor Open Source Console
Creator: Organisation: Source Auditor Inc.
Created: 2018-09-26T11:44:51Z
```

7.2.1.2 Package Information

If there is a grouping of elements (commonly files, but could be grouping of packages, etc.) to be described, then a package section should be created (see Figure 7.3). This section can be used to represent a product, a container, an upstream project source repository, or even an archive, basically any distributable component. If there are no files associated with this package in the document, then ‘Files Analyzed’ should be set to false to indicate this. By using the ‘External Reference’ field, the package can be linked to security information as well as to public repositories, in addition to any ‘Package Download Location’ provided.

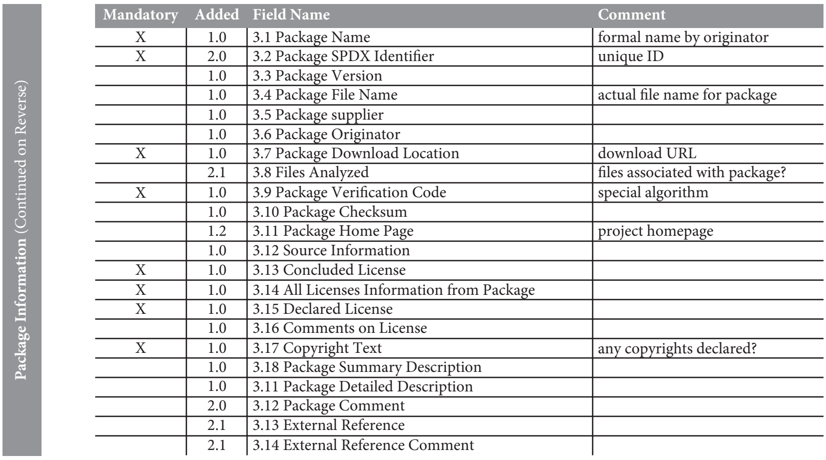
There are three mandatory fields associated with describing licensing of the package. The ‘Concluded License’ is filled in by the creator after looking at ‘All License Information from Package’ and ‘Declared License’ information. As an example, the Zephyr project sources[9] are primarily Apache-2.0 but include some files under BSD-3-Clause. So a binary built from the Zephyr project source code would have a ‘Concluded License’ of ‘Apache-2.0 AND BSD-3-Clause’, the ‘Declared License’ would probably be ‘Apache-2.0’ based on the contents of the LICENSE file,[10] and ‘All License Information from Package’ would include lines for both

[7] <https://spdx.org/licenses/CC0-1.0.html> accessed 22 June 2022.

[8] <https://spdx.github.io/spdx-spec/document-creation-information/> accessed 22 June 2022.

[9] <https://github.com/zephyrproject-rtos/zephyr> accessed 22 June 2022.

[10] <https://github.com/zephyrproject-rtos/zephyr/blob/main/LICENSE> accessed 22 June 2022.



Package Information (Continued on Reverse)

Mandatory	Added	Field Name	Comment
X	1.0	3.1 Package Name	formal name by originator
X	2.0	3.2 Package SPDX Identifier	unique ID
	1.0	3.3 Package Version	
	1.0	3.4 Package File Name	actual file name for package
	1.0	3.5 Package supplier	
	1.0	3.6 Package Originator	
X	1.0	3.7 Package Download Location	download URL
	2.1	3.8 Files Analyzed	files associated with package?
X	1.0	3.9 Package Verification Code	special algorithm
	1.0	3.10 Package Checksum	
	1.2	3.11 Package Home Page	project homepage
	1.0	3.12 Source Information	
X	1.0	3.13 Concluded License	
X	1.0	3.14 All Licenses Information from Package	
X	1.0	3.15 Declared License	
	1.0	3.16 Comments on License	
X	1.0	3.17 Copyright Text	any copyrights declared?
	1.0	3.18 Package Summary Description	
	1.0	3.11 Package Detailed Description	
	2.0	3.12 Package Comment	
	2.1	3.13 External Reference	
	2.1	3.14 External Reference Comment	

Figure 7.3 Package Information

Apache-2.0 and for BSD-3-Clause. For all licensing fields, if the SPDX document creator does not know (or does not wish to state) the applicable licence, the NOASSERTION term can be used.

Details of each field, rationale for the field, and parsing guidance can be found at the website in the footnote.[11]

An example of a package expressed as tag:value is:

```
PackageName: GNU Time
SPDXID: SPDXRef-1
PackageVersion: 1.7
PackageFileName: time-1.7.tar.gz
PackageSupplier: Organisation: GNU
PackageOriginator: Organisation: GNU
PackageDownloadLocation: https://ftp.gnu.org/gnu/time/
PackageVerificationCode: dd5cf0b17bfef4284c6c22471b277de7beac407c
PackageChecksum: SHA1: dde0c28c7426960736933f3e763320680356cc6a
PackageLicenseConcluded: GPL-2.0+
PackageLicenseInfoFromFiles: GPL-2.0+
PackageLicenseInfoFromFiles: MIT
```

[11] <https://spdx.github.io/spdx-spec/package-information/> accessed 22 June 2022.

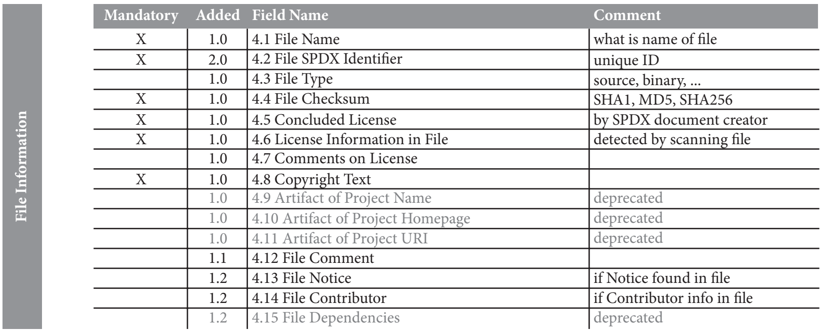
```
PackageLicenseInfoFromFiles: GPL-2.0
PackageLicenseDeclared: GPL-2.0+
PackageCopyrightText: <text>Copyright (C) 1990, 91, 92, 93, 96 Free
    Software Foundation, Inc.</text>
PackageSummary: <text>The `time' command runs another program,
    then displays information about the resources used by that pro-
    gram, collected by the system while the program was running.
    </text>
PackageDescription: <text>The `time' command runs another program,
    then displays information about the resources used by that pro-
    gram, collected by the system while the program was running. You
    can select which information is reported and the format in which
    it is shown, or have `time' save the information in a file in-
    stead of displaying it on the screen.</text>
```

7.2.1.3 File Information

Each individual file to be summarised must have a name and a checksum associated with it (see Figure 7.4).

If there is any ‘License Information in File’, then it should be documented either by an ID from the SPDX License List or via a ‘LicenseRef-’ for licences not on the list (see ‘Other-Licensing-Information’ section). In some cases, the information found in the file may not be the ‘Concluded License’ for that file, and so a second mandatory field is expected. If there is any copyright notice in the file it should also be included.

In the above table, some fields are marked as deprecated and should not be used, however they were present in prior versions of this section.



File Information

Mandatory	Added	Field Name	Comment
X	1.0	4.1 File Name	what is name of file
X	2.0	4.2 File SPDX Identifier	unique ID
	1.0	4.3 File Type	source, binary, ...
X	1.0	4.4 File Checksum	SHA1, MD5, SHA256
X	1.0	4.5 Concluded License	by SPDX document creator
X	1.0	4.6 License Information in File	detected by scanning file
	1.0	4.7 Comments on License	
X	1.0	4.8 Copyright Text	
	1.0	4.9 Artifact of Project Name	deprecated
	1.0	4.10 Artifact of Project Homepage	deprecated
	1.0	4.11 Artifact of Project URI	deprecated
	1.1	4.12 File Comment	
	1.2	4.13 File Notice	if Notice found in file
	1.2	4.14 File Contributor	if Contributor info in file
	1.2	4.15 File Dependencies	deprecated

Figure 7.4 File Information

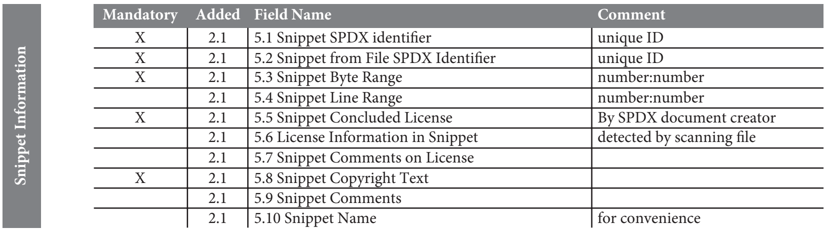
Details of each field, rationale for the field, and parsing guidance can be found in the website in the footnote.[12]

An example of a file expressed as tag:value is:

```
FileName: ./time.c
SPDXID: SPDXRef-4
FileType: SOURCE
FileChecksum: SHA1: 712d7f9dfde674283596ae2088550e3ff23ae1ba
LicenseConcluded: GPL-2.0+
LicenseInfoInFile: NOASSERTION
FileCopyrightText: <text>Copyright Free Software Foundation, Inc</text>
```

7.2.1.4 Snippet Information

Each instance of 'Snippet Information' needs to be associated with a specific 'File Information' section in an SPDX document via the File's 'SPDX Identifier' (see Figure 7.5). The 'Snippet Byte Range' field is used to identify the part of the file being described. The 'Snippet Concluded License' and any 'Snippet Copyright Text' are also required to be documented when a snippet section is used, though they can be filled in with NOASSERTION as with packages and files.


Snippet Information

Mandatory	Added	Field Name	Comment
X	2.1	5.1 Snippet SPDX identifier	unique ID
X	2.1	5.2 Snippet from File SPDX Identifier	unique ID
X	2.1	5.3 Snippet Byte Range	number:number
	2.1	5.4 Snippet Line Range	number:number
X	2.1	5.5 Snippet Concluded License	By SPDX document creator
	2.1	5.6 License Information in Snippet	detected by scanning file
	2.1	5.7 Snippet Comments on License	
X	2.1	5.8 Snippet Copyright Text	
	2.1	5.9 Snippet Comments	
	2.1	5.10 Snippet Name	for convenience

Figure 7.5 Snippet Information

Details of each field, rationale for the field, and parsing guidance can be found at website noted in the footnote.[13]

An example of a snippet expressed as tag:value is:

```
SnippetSPDXID: SPDXRef-5
SnippetFromFileSPDXID: SPDXRef-2
SnippetByteRange: 889:9002
SnippetLineRange: 24:245
SnippetLicenseConcluded: Apache-2.0
```

[12] <https://spdx.github.io/spdx-spec/file-information/> accessed 22 June 2022.

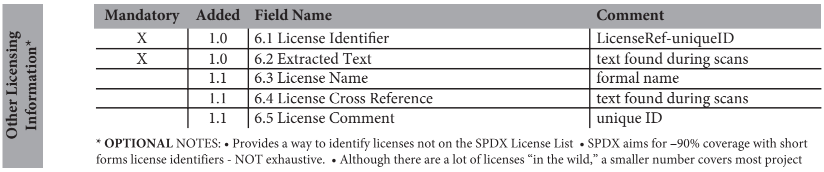
[13] <https://spdx.github.io/spdx-spec/snippet-information/> accessed 22 June 2022.

```
LicenseInfoInSnippet: BSD-2-Clause-FreeBSD
SnippetCopyrightText: <text>Copyright 2001-2016 The Apache Software
   Foundation</text>
SnippetComment: <text> This snippet should have a related package
   with an external referenced, however, the maven-plugin only sup-
   ports external references for the main package </text>
SnippetName: Apache Commons Math v. 3.6.1
```

7.2.1.5 Other Licensing Information

One instance of 'Other Licensing Information' should be created for every unique license or licensing information reference detected in the files or packages described in the document that does NOT match one of the licenses on the SPDX License List (see Figure 7.6).[14]

Each found license documented must have a 'License Identifier' assigned to the verbatim 'Extracted Text' found. The 'License Identifier' is required to start with the prefix 'LicenseRef-' to help identify it in the rest of the document. In some cases, the extracted license may have a formal name in other contexts, and the 'License Name' is an optional field to permit recording this if known.



Other Licensing Information*

Mandatory	Added	Field Name	Comment
X	1.0	6.1 License Identifier	LicenseRef-uniqueID
X	1.0	6.2 Extracted Text	text found during scans
	1.1	6.3 License Name	formal name
	1.1	6.4 License Cross Reference	text found during scans
	1.1	6.5 License Comment	unique ID

* **OPTIONAL** NOTES: • Provides a way to identify licenses not on the SPDX License List • SPDX aims for –90% coverage with short forms license identifiers - NOT exhaustive. • Although there are a lot of licenses "in the wild," a smaller number covers most project

Figure 7.6 Other Licensing Information

Details of each field, rationale for the field, and parsing guidance can be found at the website noted in the footnote.[15]

An example of an extracted licence expressed as tag:value is:

```
LicenseID: LicenseRef-FaustProprietary
ExtractedText: <text>FAUST, INC. PROPRIETARY LICENSE:
FAUST, INC. grants you a non-exclusive right to use, modify, and
distribute the file provided that (a) you distribute all copies and/
or modifications of this file, whether in source or binary form,
under the same license, and (b) you hereby irrevocably transfer and
assign the ownership of your soul to Faust, Inc. In the event the
```

[14] <https://spdx.org/licenses/> accessed 22 June 2022.

[15] <https://spdx.github.io/spdx-spec/other-licensing-information-detected/> accessed 22 June 2022.

```
fair market value of your soul is less than $100 US, you agree to
compensate Faust, Inc. for the difference.Copyright (C) 2016 Faust
Inc. All, and I mean ALL, rights are reserved.</text>
LicenseName: Faust (really) Proprietary License
LicenseComment: <text>This license was extracted from the file
InsufficientKarmaException</text>
```

7.2.1.6 Relationships

This field can be used to provide information about the relationship between two SPDX specification elements. For example, you can represent a relationship between Snippets, Files, Packages, or SPDX documents.

The relationships between two elements that are supported are:

- DESCRIBES, DESCRIBED_BY
- CONTAINS, CONTAINED_BY
- GENERATES, GENERATED_FROM
- ANCESTOR_OF, DESCENDANT_OF
- VARIANT_OF, COPY_OF
- DISTRIBUTION_ARTIFACT, PATCH_FOR, PATCH_APPLIED
- FILE_ADDED, FILE_DELETED, FILE_MODIFIED
- EXPANDED_FROM_ARCHIVE
- DYNAMIC_LINK, STATIC_LINK
- DATA_FILE_OF, TEST_CASE_OF, BUILD_TOOL_OF, DOCUMENTATION_OF
- OPTIONAL_COMPONENT_OF, METAFILE_OF, PACKAGE_OF
- AMENDS
- PREREQUISITE_FOR, HAS_PREREQUISITE
- OTHER

This set of relationships was determined by examining common use cases in the supply chain. Others can be added if a use case can be shown not to be able to be represented with the current set by opening a new issue against the SPDX specification.[16]

A detailed description and examples of each relationship can be found at the website noted in the footnote.[17]

A Relationship would follow a file or package section, and may have a comment associated with it:

```
Relationship: SPDXRef-2 PREREQUISITE_FOR SPDXRef-1
RelationshipComment: <text>The package foo.tgz is a prerequisite
for building the executable bar.</text>
```

[16] <https://github.com/spdx/spdx-spec/issues> accessed 22 June 2022.

[17] <https://spdx.github.io/spdx-spec/relationships-between-SPDX-elements/> accessed 22 June 2022.

	Mandatory	Added	Field Name	Comment
Annotations*	X	2.0	8.1 Annotator	the person, company, or tool which provided the annotation
	X	2.0	8.2 Annotation Date	
	X	2.0	8.3 Annotation Type	reviewer or other
	X	2.0	8.4 SPDX Identifier Reference	unique ID
	X	2.0	8.5 Annotation Comment	free form information

Figure 7.7 Annotations

7.2.1.7 Annotations

This section permits a person, organisation, or tool to add comments about elements in an SPDX document (see Figure 7.7). Comments can be made on snippets, files, packages, or the entire document. Annotations are usually created when someone reviews the file, but if an author wants to store extra information about one of the elements during creation, this can be used as well. If an annotation is to be made, all the sections need to be filled out. More details on the fields and values can be found at the website in the footnote.[18]

An example annotation could look like:

```
Annotator: Person: John Smith
AnnotationDate: 2018-01-29T18:30:22Z
AnnotationType: REVIEW SPDXREF: SPDXRef-5
AnnotationComment: <text>Copyright on snippet should be Copyright
2010-2012 CS Systèmes d'Information</text>
```

7.2.1.8 Specification evolution

If there is a use case you're not sure how to represent with the specification, you are encouraged to contact the volunteers at spdx-tech@lists.spdx.org and ask about it, or if you prefer, open an issue in the spdx-specification github repo.[19] If a community member can't figure out a solution, the use case will be added to the topics for the specification team to address in future revisions. As illustrated by the publishing history, this is a living specification, and continues to evolve to suit the needs of the users.

Publishing History

- 2011/08—SPDX 1.0—handles packages
- 2012/08—SPDX 1.1—fixed flaw in package verification algorithm
- 2013/10—SPDX 1.2—improved interaction with License List, additional fields for documenting project info

[18] <https://spdx.github.io/spdx-spec/annotations/> accessed 22 June 2022.

[19] <https://github.com/spdx/spdx-spec/issues> accessed 22 June 2022.

- 2015/05—SPDX 2.0—added ability to handle multiple packages, relationships between packages and files, annotations
- 2016/11—SPDX 2.1—added snippets, support for external references (CPEs, etc.)
- 2019/06—SPDX 2.1.1—move specification source to github repo to facilitate wider transparency and tracking
- 2020/05—SPDX 2.2—added SPDX Lite profile, additional support for external references (PURL, SWHid, etc.), and support for different file formats (.json, .yaml, .xml)
- 2020/07—SPDX 2.2.1—same fields as SPDX 2.2 but reformatted for ISO submission
- 2020/10—Specification submitted to ISO for balloting
- 2021/03—Balloting concludes, SPDX specification is 'Approved'
- 2021/08—SPDX specification is published as ISO/IEC 5962:2021[20]
- 2022/04—SPDX 2.2.2 published including typo fixes and clarifications

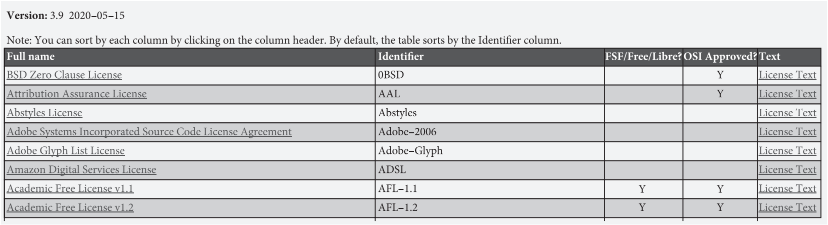
7.3 Listening to the Open Source Community Needs

The SPDX project[21] was created by developers, supply chain, security, and legal professionals collaborating with each other. This interdisciplinary team has been incrementally refining the SPDX specification (currently at version 2.2.2) and the list of recognised licenses (currently at version 3.17) over time, as the community is asked how to share specific information and licenses. If there is a use case you're not sure how to represent with the specification, you are encouraged to contact the volunteers and ask about it or open an issue against the specification.[22] If someone can't figure out a solution, the use case will be added to the topics for the specification team to address.

7.3.1 SPDX License List

The SPDX License List[23] is a list of commonly found licenses and exceptions used in Free and Open Source software and other collaborative projects, including software, documentation, hardware, data, etc. The purpose of the list is to enable easy and efficient identification of licences and exceptions and to be able to store

[20] <https://www.iso.org/standard/81870.html> accessed 22 June 2022.
[21] <https://spdx.dev/> accessed 22 June 2022.
[22] <https://spdx.github.io/spdx-spec/> accessed 22 June 2022.
[23] <https://spdx.org/licenses/> accessed 22 June 2022.



Version: 3.9 2020–05–15

Note: You can sort by each column by clicking on the column header. By default, the table sorts by the Identifier column.

Full name	Identifier	FSF/Free/Libre?	OSI Approved?	Text
BSD Zero Clause License	0BSD		Y	License Text
Attribution Assurance License	AAL		Y	License Text
Abstyles License	Abstyles			License Text
Adobe Systems Incorporated Source Code License Agreement	Adobe–2006			License Text
Adobe Glyph List License	Adobe–Glyph			License Text
Amazon Digital Services License	ADSL			License Text
Academic Free License v1.1	AFL–1.1	Y	Y	License Text
Academic Free License v1.2	AFL–1.2	Y	Y	License Text

Figure 7.8 SPDX Licence List Table

references in SPDX documents, in source files, or elsewhere. Use of these standard licence identifiers streamlines licence identification across the supply chain while reducing redundant work. They are recognised by an increasing number of upstream Open Source projects, companies, organisations, governments, and tool vendors.

The SPDX License List includes a standardised short identifier, full name, vetted licence text including matching guidelines markup as appropriate, and a canonical permanent URL for each licence and exception. When you go to the SPDX License List website, you'll see the SPDX License List table as illustrated in Figure 7.8

The first thing to note is the SPDX License List version number. It is important to keep in mind that this is a living list and gets updated approximately every three months. If you don't see a licence you are commonly encountering in Open Source code, please feel free to send a request for proposing a license or an exception to be added to the SPDX License List as directed on the webpage.[24]

The SPDX License List can also be programatically accessed as well so that the license text and matching guidelines can be used by your organisation's tools. The recommended way to get programatic access to the latest version of the License List is through the License List data project on GitHub,[25] rather than scraping the website. The repository there contains various generated data formats for the SPDX License List, including JSON, RDFa/HTML, RDF NT, RDF turtle, RDF/XML, and HTML, as well as a simple text version. More details on how to access the SPDX License List programatically can be found at the website in the footnote.[26]

[24] <https://github.com/spdx/license-list-XML/blob/master/CONTRIBUTING.md> accessed 22 June 2022.

[25] <https://github.com/spdx/license-list-data> accessed 22 June 2022.

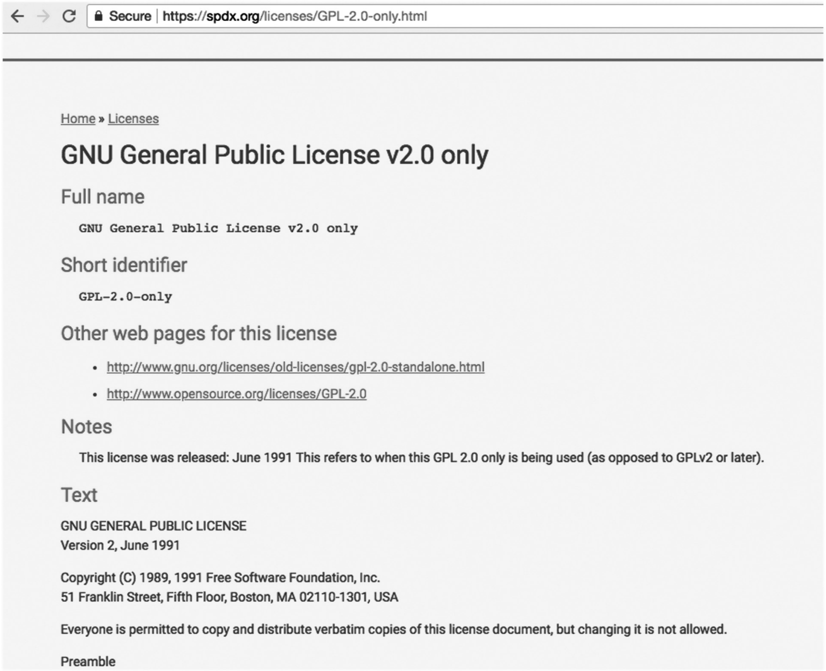
[26] <https://github.com/spdx/license-list-data/blob/master/accessingLicenses.md> accessed 22 June 2022.

In the SPDX License List table shown in Figure 7.8, you'll see the columns for:

- Full Name of the licence.
- Identifier for the licence. This 'short identifier' also gets referred to as the SPDX license ID in some places.
- FSF Free/Libre? If the licence is considered free by the Free Software Foundation (FSF), this field will indicate 'Y', otherwise it is left blank
- Is OSI Approved? If the licence is approved by the Open Source Initiative , this field will indicate 'Y', otherwise it is left blank
- A link to License Text of the license. The full text of the licence is provided as well as any standard headers associated with a licence.

If you click on the column headers it will sort the SPDX License List table by those fields.

By clicking on the 'Full Name' or the 'License Text' you'll also be taken to a canonical permanent URL for that license that provides more information about the license. The permanent URL can be found by appending the short identifier and '.html' to the 'https://spdx.org/licenses/' prefix. An example of the GPL-2.0-only license page is reproduced later in this chapter.



Secure | https://spdx.org/licenses/GPL-2.0-only.html

Home » Licenses

GNU General Public License v2.0 only

Full name

GNU General Public License v2.0 only

Short identifier

GPL-2.0-only

Other web pages for this license

- http://www.gnu.org/licenses/old-licenses/gpl-2.0-standalone.html
- http://www.opensource.org/licenses/GPL-2.0

Notes

This license was released: June 1991 This refers to when this GPL 2.0 only is being used (as opposed to GPLv2 or later).

Text

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

7.3.2 Clarifying licensing and metadata information in source code

Accurately identifying the licence for Open Source software is important for licence compliance. However, determining the licence can sometimes be difficult due to a lack of information or ambiguous information. Even when there is some licensing information present, a lack of consistent ways of expressing the licence can make automating the task of licence detection very difficult, thus requiring significant amounts of manual human effort. There are some commercial tools applying machine learning to this problem to reduce the false positives and train the licence scanners, but a better solution is to fix the problem at the upstream source.

The SPDX project liked the simplicity of this approach introduced by the U-Boot project in 2013,[27] and formally adopted the syntax for embedding 'SPDX License Identifier' tags into the project and documented the syntax in SPDX specification from version 2.1 onwards.[28]

The SPDX License Identifier syntax used with short identifiers from the SPDX License List (referred to as SPDX License IDs) can be used to indicate relevant licence information at any level, from package to the source code file level. The 'SPDX License Identifier' phrase and a license expression[29] formed of SPDX Licence IDs in a single-line comment form a precise, concise, and language-neutral way to document the licensing that is simple to machine process. This leads to source code that is easier to read, which appeals to developers, as well as enabling the licensing information to be trivially searchable via grep and to travel with the source code.

To use SPDX License IDs in your project's source code, just add a single line in the following format, tailored to your license(s) and the comment style for that file's language. For example:

```
// SPDX-License-Identifier: MIT
/* SPDX-License-Identifier: MIT OR Apache-2.0 */
# SPDX-License-Identifier: GPL-2.0-or-later
```

[27] <https://git.denx.de/?p=u-boot.git;a=commit;h=eca3aeb352c964bdb28b8e191d6326370245e03f> accessed 22 June 2022.

[28] <https://spdx.github.io/spdx-spec/using-SPDX-short-identifiers-in-source-files/> accessed June 2022.

[29] <https://spdx.dev/ids> accessed 14 April 2022.

To learn more about how to use SPDX License IDs with your source code, please see the documentation in the SPDX project,[30] and David Wheeler's tutorial.[31]

The use of these short identifiers to identify the licences has been adopted by other upstream Open Source projects and repositories, including GitHub in its licences' application programing interface (API).[32] In addition to U-boot, Linux is transitioning to use the SPDX License IDs, and newer projects like Zephyr and Hyperledger Fabric have adopted them right from the start as a best practice.[33] Indeed, to achieve the Core Infrastructure Initiative's gold badge, each file in the source code must have a licence, and the recommended way is to use an SPDX License ID.[34]

> `The project MUST include a license statement in each source file. This MAY be done by including the following inside a comment near the beginning of each file: SPDX-License-Identifier: [SPDX license expression for project].`

When SPDX License IDs are used, gathering license information across your project files can start to become as easy as running 'grep'. If a source file gets reused in a different package, the licence information travels with the source, reducing the risk of licence identification errors, and making licence compliance in the recipient project easier. By using SPDX License IDs in licence expressions, the meaning of licence combinations is understood more accurately. Stating 'this file is MPL/MIT' is ambiguous and leaves recipients unclear about their compliance requirements. Stating 'MPL-2.0 AND MIT' or 'MPL-2.0 OR MIT' specifies precisely whether the licensee must comply with both licence, or either licence, when redistributing the file.

As illustrated by the transition underway in the Linux kernel,[35] SPDX License IDs can be adopted gradually. You can start by adding SPDX License IDs to new files without changing anything already present in your codebase.

[30] <https://spdx.dev/ids-how> accessed 22 June 2022.

[31] <https://github.com/david-a-wheeler/spdx-tutorial> accessed 22 June 2022.

[32] <https://developer.github.com/v3/licenses/> accessed 22 June 2022.

[33] <https://spdx.dev/ids-where> accessed 22 June 2022.

[34] <https://github.com/coreinfrastructure/best-practices-badge/blob/master/doc/other.md#basics-1> accessed 22 June 2022.

[35] <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/process/license-rules.rst> accessed 22 June 2022.

7.4 Tooling and Best Practices to Make it Easy for Developers

In 2017, the Free Software Foundation Europe (FSFE) created a project called REUSE.software[36] that provided guidance for Open Source projects on how to apply the SPDX License Identifiers into projects. The REUSE.software guidelines were followed for adding SPDX License Identifiers into the Linux kernel later that year.[37] In addition to the guidelines propose by the REUSE.software project, there is also a linter tool that can generate an SBOM automatically, if the guidelines are followed.[38]

The SPDX project also maintains a set of Java-,[39] Python-,[40] and Go-[41] based tools to help with validation of SPDX documents and conversion between the supported file types. These libraries are available for other tool creators to use, and can simplify the creation and consumption of SPDX documents with their software. As with the Specification and License List, suggestions for improvement to the SPDX tools are also appreciated.

7.5 Adoption of SPDX Documents

The use of SPDX as a recognised SBOM format has been slowly improving over the last six years. A key factor was the introduction of the open source scanning tool FOSSology[42] being able to output SPDX documents in 2016, and then adding the capability to consume them in 2019. Before then, the only scanning tools able to work with the format were proprietary. Other Open Source tools have since become available to help with specific workflows, and this is accelerating adoption.

Another key factor was the growing adoption of the OpenChain Specification that calls for the ability to create a Bill of Material (BOM) for the software in its conformance criteria.[43] Following on from OpenChain becoming an ISO standard

[36] <https://reuse.software/> accessed 22 June 2022.

[37] <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/process/license-rules.rst> accessed 22 June 2022.

[38] <https://github.com/fsfe/reuse-tool> accessed 22 June 2022.

[39] <https://github.com/spdx/tools/> accessed 22 June 2022.

[40] <https://github.com/spdx/tools-python> accessed 22 June 2022.

[41] <https://github.com/spdx/tools-golang> accessed 22 June 2022.

[42] <https://www.fossology.org/> accessed 22 June 2022.

[43] <https://wiki.linuxfoundation.org/_media/openchain/openchainspec-2.0.pdf> accessed 22 June 2022.

(ISO/IEC 5230:2020), as discussed in Chapter 6, procurement departments in the supply chain will be more likely to expect to see an SBOM with delivery of software and producing this will be simpler. The SPDX specification became an international standard (ISO/IEC 5962L2021) in 2021.

In June 2018, the National Telecommunications and Information Administration (NTIA) engaged stakeholders across multiple industries to discuss software transparency and determine what a minimum viable SBOM is, and what file formats could support this information. SPDX was recognised as a valid standard to support SBOMs in the wrap up of the 2019 Phase 1 work.[44] Phase 2 used SPDX to share data between medical device manufacturers and health delivery organisations.

Over the recent years, there has been a growing awareness in the industry that there is a need to improve the transparency of the software running on systems. In November 2020, the EU put out the ENISA report, 'Guidelines for Securing the Internet of Things', which calls out as a best practice to provide a SBOM for Internet of Things (IoT) devices.[45] On 12 May 2021, the US Biden Administration issued a Cybersecurity executive order, calling for a best practices in Enhancing Software Supply Chain Security (Section 4) to 'providing a purchaser a Software Bill of Materials (SBOM) for each product directly or by publishing it on a public website' (Section 4(e)(vii)).[46]

7.6 Future Directions

There are several use cases the SPDX community is considering how best to represent as it works towards the next version of the specification. From the NTIA SBOM framing working group efforts,[47] as well as the OpenChain Japan team efforts in creating the SPDX Lite profile,[48] it became clear that having a minimal base set of fields to just represent the manifest and relationships was needed. The SPDX community is working to refactor the specification into a basic set of fields (called

[44] <https://www.ntia.gov/SBOM> accessed 22 June2022.

[45] <https://www.enisa.europa.eu/publications/guidelines-for-securing-the-internet-of-things> accessed 14 April 2022.

[46] <https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity> accessed 22 June 2022.

[47] <https://www.ntia.gov/SBOM> accessed 22 June 2022.

[48] <https://spdx.github.io/spdx-spec/SPDX-Lite/> accessed 22 June 2022.

core) with optional profiles, to handle specific domain information like licensing, security, pedigree, provenance, and usage. If you have use cases in these areas, and want to participate in the discussion, please open an issue on GitHub,[49] or send mail to spdx-tech@lists.spdx.org.

[49] <https://github.com/spdx/spdx-spec/issues> accessed 22 June 2022.

8

Corporate Concerns

Audit, Valuation, and Deals

Toby Crick

8.1 Introduction

8.1.1 Open Source and corporate culture

Not so long ago there was a view amongst Chief Information Officers (CIOs), investors, and board-level executives that Open Source software (let alone free and open innovation) was an anathema to the key metrics they were measured against: maintaining value within the corporation with a view to maximising revenue and profitability and ultimately only considering shareholder or investor value.

Toby Crick, *Corporate Concerns* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press.
© Toby Crick 2022. DOI: 10.1093/oso/9780198862345.003.0008

Even as they adopted Linux-based servers and admired the business models of companies such as Red Hat, the prevailing view was that Open Source was for hobbyists and activists, not for 'proper' corporations.

8.1.2 Widespread use of Open Source

It is safe to say that this point of view is now entirely out of date. Once in a while an investor or advisor may flag a concern about Open Source but the undoubted benefits Open Source can bring to an enterprise in terms of time to market, reduced software development cost, and improved code quality is now widely understood. Indeed, the use of Open Source by enterprises in their own code development and in the third-party products they licence to use to operate their businesses is now standard practice.

As further discussed in Chapter 19 this process of use to contribution, has no doubt been helped by the wider adoption of permissive licence models (such as MIT or Apache) as discussed in Chapter 2, and means that today, the use of Open Source is seen as less likely to put at risk future monetisation opportunities and may in some cases be the heart of a monetisation opportunity.

One cannot imagine today a Chief Executive Officer (CEO), CIO, or investor mandating that 'no Open Source shall be used in this company'. Instead, they, and technology investors in particular, would be more likely to have significant concerns about an over-reliance on proprietary code than they would over the *well-managed* use of Open Source or a company that was not aware of its dependence on Open Source software.

8.1.3 Managing the use of Open Source in the enterprise

Of course, just like any other activity involving technology and software, the use of Open Source is not without risk. What is surprising is that now that the use of Open Source is standard across the corporate world, the use of sensible risk management protocols in relation to such use is not also standard. This is particularly surprising given the exponential growth of tools and processes aimed at risk management being made available to users and developers of Open Source. Examples of governance and risk support tools like the ISO standard Open Chain discussed in Chapter 6 abound and are part of the Open Source community's journey to standardised good practice.

It is clear that there is an expectation today that enterprises will utilise such tools and processes and will need to demonstrate such use to investors and regulators alike that they have an adequate Open Source policy and processes in place to manage governance and risk of their use of and, if appropriate, contribution to, Open Source software.

This chapter:

- looks at code audits and the good practices around managing and logging the enterprise's code base across proprietary third-party tools, third-party Open Source tools, and its own internally produced code (see section 8.3 later in this chapter) that an enterprise can deploy;
- then considers how Open Source can add value to a business or even become a valuable business in and of itself (see section 8.4);
- sets out how the process of mergers and acquisitions (M&A), investment, and ultimately an Initial Public Offering (IPO) should be adapted where a key asset of a target business (whether used internally or to create revenue) is Open Source; and
- investment around Open Source.

8.2 Why Understanding Open Source is Important in the Corporate Context

8.2.1 Good practice

As noted earlier and throughout this book, the ease of use and low initial cost of Open Source means that its use is now very widespread and Open Source projects have an advantage of their packages and products being able to scale quickly and without lengthy procurement processes. However, even though Open Source is 'freely' available and is generally also cost-free, there are significant issues that can impact on its commercial and legal attractiveness.

It is essential that these issues are understood and managed. There is a wide body of work describing best practice and a huge and growing ecosystem of service providers and consultants whose business it is to help enterprises manage their use of Open Source in a risk-appropriate manner.

8.2.2 Key risks

The risks associated with the use of Open Source are briefly considered here.

8.2.2.1 Copyleft and 'viral impact' of Open Source

As discussed in detail in Chapters 2 and 3, some Open Source is licensed for use on terms that require any onward distribution of such software (even in other products) to be licensed on the same terms through the mechanism of copyleft or reciprocal licences. Enterprises must be aware of this and ensure that they do not bundle copyleft code in with their commercial products in a way that could cause a viral impact, however technologists are more than familiar with the ways in which to manage this risk and that the solution to avoiding 'viral infection' (a term loathed by the Open Source development communities) is to manage linking by ensuring it is dynamic not static and to use other technological solutions such as RCP[1] in order to avoid the derivative work issue which causes such a risk. This requires suitable internal processes and flags in the use of code under such licences and is a great example of a technical solution to a legal problem.

As fully discussed in Chapter 3, the compatibility of this code with other Open Source software also needs to be considered.

8.2.2.2 Lack of IP infringement claim protection

Open Source licences do not include any IP warranty or indemnity protection in relation to IP infringement claims and indeed the well written ones specifically disclaim any implied liabilities. This is due to their free, no-cost nature.[2] When compared to proprietary software, where the licensor in theory knows the genesis of the software and can be reasonably confident of no copyright infringement (and to a lesser extent, patent infringement), we see the proprietary companies offering higher levels of indemnity around copyright. However, in recent years their willingness to take on liability for patent risk has significantly reduced, somewhat levelling the playing field. Of course, the indemnity from a proprietary vendor is provided in return for a licence fee or royalty and this is part of a calculation around the cost of software. The cost of insurance for such liability may be an alternative to the licensing fee.

Whilst single product Open Source companies and aggregators like Red Hat may offer some level of protection with subscription, upstream licensors of Open Source are unlikely to provide any such comfort.

This means licensees/users of Open Source are potentially exposed to third-party IP infringement claims with no recourse against the upstream licensors (who may be an individual uploading code via GitHub or a large corporation). However, the value of a corporate indemnity from a proprietary vendor is as good as the vendor or their insurer. In this respect use of insurance for Open Source is a way to level the playing field if this is a particular concern to a user or licensor of Open Source software.

[1] A Rich Client Platform (RCP) is a computer program allowing the creation of Java applications in a portable and reusable manner.

[2] If, however, Open Source is provided by a commercial supplier with add on subscription or services (e.g. from a vendor like Red Hat or Tidelift), then a level of indemnity may be provided as part of this service, similar to the indemnities provided by proprietary software providers. Even here such indemnity protection is unlikely to include IP infringement claims.

The financial and reputational impact of being found liable in, or even just having to defend, an IP infringement claim will be disruptive for a business. When coupled with the potential loss of the right to use a business-critical program or Open Source which underpins a significant revenue stream, it is crucial that this risk is considered when managing or investing in an enterprise that uses Open Source and good practices such as the use of Open Chain are followed to alleviate risk.

While the risk of an IP infringement claim exists, the actual risk of such a claim is, in practice, low.

8.2.2.3 Lack of performance warranties or support and maintenance

If an enterprise downloads Open Source from the web it will have a right to use, modify, etc. in accordance with the licence, but without a paid relationship with a provider, the enterprise will have no liability protection around the quality or fitness for purpose of the Open Source. While in practice the risk of an IP infringement claim is low, the risk of an issue arising with code or the way the code interoperates with other systems used by the enterprise is much more real and requires internal skills in integrating the code in an appropriate way and in assessing the suitability of the code. Open Source software ecosystems have been quick to build and are extensive around certain code like Linux and Kubernetes but an enterprise will need to have knowledge of such ecosystems and the skills to use them to benefit from the risk mitigation they offer.

8.2.2.4 Scalability and robustness

In a similar vein, if an enterprise opts to download and use Open Source, it becomes responsible for ensuring that the code is robust and reliable and its use can scale up to meet enterprise level requirements and any requirements in the service levels of contracts with its own customers.

8.2.2.5 Security vulnerabilities

All software is at risk of being hacked and of security breaches as is more fully discussed at Chapter 13. Open Source systems are potentially more at risk because their code is free and open but at the same time may be better protected and fixes may be easier (and cheaper) to apply. Their open nature does indeed mean that anyone can look at it and improve it but also anyone can look at it, find vulnerabilities, and exploit them.

Given the fact that (unless you sign up to obtain the same from a specialist vendor) there is no software vendor pushing out security patches to users and rather the user is dependent on the ecosystem of maintainers for a project, a well-led IT department will monitor and manage its Open Source estate for vulnerabilities. Part of this process will include considering maintenance and management of security vulnerabilities in an Open Source project in its choice of enterprise usage of

Open Source and its assessment of individual packages it chooses to bring into its business, and its ongoing assessment for vulnerabilities. At a fundamental level for all software, not just Open Source software, a well-run IT department will ensure that if a patch is made available (e.g. for Open Source, on GitHub) it is applied.[3]

This has been reflected in a number of recent moves to improve maintenance of critical Open Source projects by organisations like the Linux Foundation and in commercial models offering single vendor aggregation and packaging of Open Source software by companies like Tidelift.

Of course, security is also identified as a reason to move from simply being a user of Open Source to being a participant in key projects contributing and gaining the ability to help shape the development of the product features.

8.2.2.6 Summary

An understanding of the commercial risks of using Open Source is often the starting point of an organisation's engagement with Open Source and is the reason that initial engagement with Open Source may come from the legal department. As we will see, there are a range of steps a well-advised enterprise can take to manage these risks and in building its policies and practices and if it evolves in its Open Source journey to building an Open Source Program Office and these are more fully covered in Chapter 19.

8.3 Open Source Audit Services

8.3.1 The context within which Open Source audits take place

Best practice is, of course, to ensure that all code used in an enterprise is logged and managed as it is created, licensed in from a software vendor, or downloaded from an Open Source repository.[4] However, many (possibly most) organisations do not do this and even those that do need to test their own logs against 'reality' to ensure what they have logged is indeed what they use.

This is crucial in the context of managing the security of an enterprise's systems but takes on particular importance when an enterprise is being acquired, going through an Initial Public Offering (IPO) or is about to receive significant investment.

[3] It is alleged that hackers got into 'Panama Papers' law firm Mossack Fonseca's systems via its Open Source website and client portals. The alleged issue was not that those systems had vulnerabilities but that Mossack Fonseca had not updated the systems to fix known security vulnerabilities: <https://www.theregister.co.uk/2016/04/07/panama_papers_unpatched_wordpress_drupal/> accessed 14 April 2022.

[4] Clearly, enterprises whose business is the creation and licensing of Open Source code are much more likely to maintain an up-to-date code base than those that simply use software to enable their enterprise to operate.

In such circumstances even an enterprise with an incredibly well-managed code base is likely to need to undertake a software audit to demonstrate this and to supply a contractual commitment to its investors or as part of its disclosure documentation[5]. For most enterprises—particularly those with perhaps less robust systems—a code audit may well start from scratch with little or anything to go on in terms of existing code logs.

8.3.2 Purpose of Open Source audit

While one aim of an Open Source audit will be to verify that the enterprise's own records are up to date and correct, generally speaking the aim of the audit is to:

- identify the Open Source used by the enterprise;
- identify the licence terms applicable to the code used;
- identify how the Open Source is used and interactions with other packages (e.g. is it only used internally or is packaged up into solutions sold, or otherwise made available to customers, by the enterprise);
- identify whether and to what extent matters such as security patching and other support and maintenance activities are undertaken in relation to the Open Source code used by the enterprise;
- identify any code licensed out by the enterprise on Open Source terms (and if so, which terms are applied).

Enterprises seeking investment, IPO, or sale would be well advised to carry out internal audits before beginning an investment round, IPO, or sale process as then they will be prepared for the inevitable investigation an investor or acquirer may undertake. Ultimately, the aim of an audit is to assess risk and consider how well-managed the enterprise's Open Source estate is.

8.3.3 Audit process

There are two main approaches to audit.

8.3.3.1 Automated audit

As the title implies, this is a process whereby the enterprise's code base is exposed to a software tool that 'crawls' over it and compares the code line by line to an inventory of Open Source the tool provider maintains. There are a number of

[5] 'Disclosure' is the process, often undertaken in an M&A transaction, whereby a target shares information about itself and its assets with a potential purchaser; see section 8.5.1 later in this chapter.

commercial providers, the most well-known being Flexera and Black Duck. Other services, including free, Open Source-based ones such as Open Sourceology, are available.

Typically the automated service is used as part of a regular update and review process undertaken within an enterprise. In the context of a major piece of M&A or an investment—particularly into a technology or software business—the automated tools would be supplemented by a human-led expert audit.

It is worth noting that an audit tool can only ever be as good as its code base, in other words the code base that it is run against.

8.3.3.2 Expert audit

This is a human process, where experts review the enterprise's use of software and its logs and other records. This typically also involves interviews with developers and other members of the IT department.

Expert audits would also need to involve an automated 'code crawling' audit. The expert audit team then use the output of that automated audit to inform their report and focus their further investigations and enquiries. Unlike where an internal IT team or a mainstream advisor or deal team member from an investor or acquirer reviews an automated audit report, the expert audit team would generally be better able to spot issues in an automated report or make further enquiries rather than just rely on the first output of the automated report.

The expert auditors can also form a view on the wider health of the way in which an enterprise manages its code and the related risks. Often the first thing they will want to see is the internal policies and Open Source logs (i.e. as referenced in section 8.2.3 earlier in this chapter). In the context of M&A and investments, a good report from an auditor can be a key piece of due diligence for the target company while a bad report can be of crucial importance to the incoming owners/investors. At worst, a poor audit report can prevent a deal from happening but more often it reveals issues that the target must address in order for the sale to proceed.

Finally, it is worth noting in the context of expert audits that whilst there is still a place for this the increasing scale of the number of packages used, sometimes in their thousands, has reduced the possibility of a manual, line-by-line code review and fix by audit teams.

8.3.3.3 Pros and cons

In simple terms, the automated reviews are good insofar as they go. They change over time and different tools have different strengths and weaknesses, but collectively none are perfect and the tools cannot in and of themselves show how robust or well run an enterprise's technology is.

They have to be interpreted and for regular internal audits that is probably fine. The enterprise's team can use the output to inform their investments and strategies.

However, for a major investment (either preparing for one or undertaking one), investing in a full expert audit probably does make sense.

Of course, the major downside of an expert audit is cost. They are expensive and, for a simple business with either little reliance on Open Source outside the enterprise or little use of Open Source, they may not be needed.

A well-managed Open Source policy and processes being in place in an organisation in advance of this clearly significantly reduces the burden of this process. It is also worth noting that where issues arise in an audit, in the main the fixes are technical, not legal, and begin by considering the removal of any unused code that has been flagged as an issue that may not be used, swapping problem code and then building technical fixes or workarounds.

8.4 Valuation

8.4.1 How technology assets can contribute to the valuation of a business

Generally a business is worth what someone is willing to pay for it based on cash flows, profitability, etc., but sometimes just the 'technology assets' of a business give rise to its valuation. In other words, an investor or acquirer of an enterprise may consider that the target's products have such potential either to enhance the value of the investor or acquirer's other technologies or, marketed or 'monetised' differently, could achieve a value on their own such that it is worth investing in them.

8.4.2 How can a business reliant on something distributed for free be given a value?

If an enterprise uses a suite of third-party Open Source components to run its operations ultra-efficiently but does not derive revenue from making its tools available to third parties, then clearly the value of its technology to it (and any future group it may be acquired by) relates to its ability to drive value from the business, not from the software itself. In such instances, investors will be seeking to ensure that the Open Source estate is used lawfully and prudently but they will not have the same focus on identifying whether the Open Source *in itself* can deliver value to the business (and them, as investors).

8.4.2.1 Valuation where Open Source is core to the business's revenue and growth prospects

Where the creation and distribution of Open Source is core to a business then how it is valued becomes a key issue that any investor (or entity seeking investment) must consider. As discussed in Chapters 2, 15, and 16, the answer to the question posed in the heading above is that there are a range of ways to identify revenue streams that a business that has Open Source at its core can take advantage of.

When it comes to assessing value in the context of an investment round, an M&A deal, or an IPO, interested parties need to consider the business' Open Source assets, its knowledge, skills, and knowhow related to the use and exploitation of those assets, and—and this is fundamental—the viability of its business model.

A valuation is based on the return an investor (or buyer) expects to get from its investment. Assuming that a business has secured and managed its Open Source assets and knowhow, then the key to valuation becomes what the existing management or the management of a business looking to acquire an Open Source-based business thinks it can generate.

Here, care is needed. There have been a number of examples of hugely successful IPOs or trade sales of Open Source businesses (notably Github's sale to Microsoft and RedHat's to IBM), but in other cases businesses backed by private equity investors seeking a rapid scale up in revenue and, above all, profitability have given rise to issues of underperformance (at least in relation to the expectations investors had of the business) and licence changes.

When assessing the value of an Open Source-based business, its ability to grow revenues and maintain long-term profitability should be carefully assessed. The metrics traditionally applied to proprietary technology businesses may not work and may be more attuned to an Open Core-type model (see Chapter 16).[6]

Clearly the more well established an Open Source business's commercial offering is, the easier it is to ascribe a value to it. Even here though, care is needed as the ability of a business to grow revenue or to turn healthy revenues that are also matched by high costs (e.g. in investing in and maintaining a code base) must be considered by an investor or acquirer. There is of course also a question of whether there is a need to update the approach to valuation of Open Source-based businesses.

8.4.3 How ill-managed Open Source assets can undermine the valuation of a business for IPO, acquisition, and investment

In this context it is worth considering the counterfactual: where poor use of Open Source can undermine a valuation. Consider, for example, a situation where a company is licensing a product on a proprietary basis but has, in fact, packaged up and compiled a range of truly copyleft Open Source components and, either due to

[6] The concept of open core is akin to freemium type offerings in the AppStore or GooglePlay whereby a core product is made available on Open Source terms but additional functionality is licensed, and paid for, on a proprietary software model.

ignorance or as blatant fraud, failed to declare this when it on-licences them to its customers.

Potentially that company's entire revenue stream was earned by breaching the terms of the copyleft licences it was party to in creating its own code's core components. In such situations investors or acquirers may either walk away from the deal, revalue the deal, or require the sellers to invest substantial sums in reworking their code and put sale proceeds at risk should their original mistake in wrongly commercialising copyleft code come to light.

8.5 Issues Arising on M&A

There are two main situations where Open Source will need to be considered in an M&A context. The most obvious is where the target is a technology business (i.e. generates revenue and profits from the development and sale or licensing of software or other technology). However, since all businesses use technology and most in a digital world both license software in and generate their own for internal use, in all but the most non-technology deals, how any business uses Open Source will still need to be considered in an M&A context.

8.5.1 M&A process

While the role of this chapter (let alone this book) is not to set out in detail how M&A transactions (or investments or IPOs) work, in order to consider how to manage Open Source issues in the M&A context it is probably worth a quick summary of the M&A deal process.

Typically, a company will either put itself up for sale or be approached by a potential acquirer. In the latter case the parties then tend to agree an outline of a proposed deal or *Heads of Terms* before the buyer engages in process of investigation known as *Due Diligence*. In the former case the seller may provide initial due diligence information to potential bidders before moving to Heads of Terms and full Due Diligence with an agreed bidder moving to contract. There are of course many different permutations (before you even consider contested public company takeovers), but this process of agreeing an outline deal and then carrying out investigations to make sure the deal is worth the money is the standard approach established over decades.

In parallel to the diligence process the parties will negotiate a substantive sale and purchase agreement or contract. This will, amongst other things, set out a series of *Warranties* (effectively promises) from the seller about the health and wellbeing of its business. Alongside these warranties the seller will make *Disclosures* effectively stating that whilst, for example, it warrants that all of its systems and procedures comply with whatever standard the buyer wants, in certain respects it has fallen short.

How strong the warranties provided are and how broad the disclosures are becomes a key area of negotiation between the parties. This is due to the fact that ultimately, if the deal closes and the buyer discovers that all is not well with its newly acquired business, it will rely on the warranties to seek recompense from the seller (i.e. the former owners of the company) and the seller will seek to avoid those claims by stating it made full disclosures and that the buyer 'knew what it was getting'.

It is worth noting that under English law, warranties, indemnities, and conditions have very established meanings and consequences set out by decades of case law and the actual meanings of these and consequences of their inclusion can be somewhat different for the US and the UK (and different again in other jurisdictions), yet the inclusion of these and their high-level purpose is the same in most markets and jurisdictions, albeit local law principles must be applied on top.

In England, a breach of a warranty will allow the recipient to claim damages and a breach of a condition may allow for the contract to be set aside. English judges reserve the right to decide whether something is a warranty or condition, regardless of what it is described as in a contract. Generally in M&A deals, the contract makes clear which is intended to be which and where remedies are only financial or can lead to the complete reversal of the transaction (the latter being very rare).

In many cases warranties will be backed by indemnities, which are statements of what a party's liability will be and what for, or a promise that if something either happens or does not happen an amount will be paid. The word indemnity has no magic meaning and the value of an indemnity will depend on what it actually says and commits the party granting it to.

8.5.2 Importance of Open Source

As noted earlier, how important Open Source is to an M&A deal depends on the business—if the business sells software or services based on it, then understanding the code base is crucial; if the business is a tech-enabled business then understanding its code base is also crucial; but if it is a normal digital business that might use some Open Source in the back office or infrastructure, then it may be less so.

At each stage of the deal—Heads of Terms, Due Diligence, in the sale and purchase agreement's Warranties and the linked Disclosures—Open Source will need to be considered.

8.5.2.1 Issues related to Open Source

8.5.2.1.1 Head of terms

In any technology-related M&A deal the Heads of Terms should recognise the existence of Open Source and call out the level of its importance to the business being acquired. How much detail the Heads of Terms go into will depend on the business and the Open Source it uses but typically you would expect the Heads of Terms to state that there will be a process to verify the importance of Open Source and what Open Source there is.

At this stage both parties should consider if there will be an Open Source audit and if it is possible to obtain the results before full diligence starts in order to avoid nasty surprises (and unnecessary costs down the line). Where a company that is reliant on Open Source puts itself up for sale it would be common for it to have undertaken internal audits prior to starting the sale process in order to ensure that its code base is 'clean' and its procedures (even if this involves some reverse engineering) are in line with good practice.

It is typically prudent to identify early if the Open Source used by the target includes true copyleft material and in particular whether it is being distributed as part of the target's activities and if so, what impact that has on deal. The impact may be that there is no deal, hence the importance for a company seeking to sell itself to carry out internal audits before starting a sale process.

8.5.2.1.2 Due diligence

During the due diligence process the buyer will seek to gain as much information as it can on areas it sees as particularly risky. Where the target relies on Open Source the key questions to ask are to see a list of all Open Source components, to seek confirmation (either by automated or full expert audit) that such list is complete, and to see the target's Open Source use and management policies, and to gain an understanding of how well such policies are observed. Where the seller/target is a software company, it might share the output of its own software audit but a purchaser might want to appoint a third-party auditor to undertake an automated or possibly automated plus expert audit.

At section 8.3.1 earlier the importance of code logs is discussed; it goes without saying that if the diligence process reveals that an enterprise's code logs are up to date and correct this will be a big diligence 'win' and give confidence to an investor or acquirer. Conversely if it shows the logs are wrong or out of date this will indicate a culture of poor housekeeping and undermanaged risk control procedures and may lead to the deal unravelling, the price dropping, or the warranties being tightened up considerably.

Where the deal involves competitors then the target may well want to limit how much information it shares (and of course if competition law/anti-trust issues may arise, both parties will have to be careful as to how much information is shared). This is where a third-party audit can be helpful. The third party can identify key risks and issues but not share any details of code or product strategies with the buyer.

8.5.2.1.3 Warranties

Where an audit has been undertaken there is likely to be a requirement for a warranty around that audit. It can only ever be as wide as the provider of the audit, whose contract terms are likely to take little liability for the accuracy. An audit warranty will generally refer to the audit being undertaken by the agreed provider and state that identified issues have been rectified and be accompanied by a disclosure of the final signed off audit document or a link to the code base, depending on scale.

It would now be unusual for a buyer to seek a warranty in an agreement stating that the target did not use Open Source.

However, it remains common to see warranties that:

- There are 'no copyleft' components used in materials distributed to customers—here the seller may try and ensure that it has an exception where they are distributed in a way that does not put its core product at risk of also being deemed copyleft, and this may be a simple technical structuring issue;
- there is 'no Open Source other than as disclosed' used in the business—here the seller may want to try and apply a materiality threshold or link to an audit report and this should be qualified by reference to the audit provider; and
- any Open Source materials licensed to third parties are licensed 'in full compliance with the terms of the Open Source licences granted to the seller' and 'all licence requirements are complied with in relation to the distribution of Open Source software'—while a seemingly simple commitment, many businesses struggle to commit to this warranty fully. A business' ability to stand behind this warranty will require a business to have in place sensible policies and compliance procedures to manage the process of licence compliance in relation to the utilisation of the code (e.g. by ensuring attribution in headers and making source code available). Again, a technical requirement or solution emphasising the need for the legal people working on such warranties to work closely with the technical team in ensuring compliance with the warranties offered is key to a business's ability to sign up to this type of warranty in a sale and purchase (or investment) agreement.

8.5.2.1.4 Disclaimer of warranties

In software licences, it has increasingly become market practice for the licensor to disclaim any IP or performance warranties (and exclude from third-party IP infringement indemnities) any issues related to third-party Open Source components. Instead the licence will state that such components are provided 'as is' and/or oblige a customer to take responsibility for downloading the Open Source components required to make a solution work on its own account. This is considered further in Chapters 3 and 16.

It should be clearly understood that this type of disclaimer is much harder to obtain in an M&A context. If, for example, the target distributes Open Source-based products, both the vendor and buyer will need to undertake careful diligence and a detailed negotiation and disclosure process to understand and allocate the risks that using Open Source—particularly undocumented use of Open Source—may give rise to, and businesses with a significant reliance on Open Source software may need to work to ensure that their counterparties understand the inappropriateness of wide-ranging warranties and why there is a need for warranty disclaimer from their acquirers and investors.

8.5.2.1.5 Disclosures

The process of disclosure is the seller's opportunity to avoid liability for breach of warranty by formally disclosing any breaches it is aware of. This is where the diligence the parties have undertaken comes into its own.

Where the seller is selling a software business (or a software-rich business that is active in other sectors), it would be prudent to undertake an audit in order to be aware of its Open Source risks and to disclose against them. Of course, complying with good Open Source housekeeping principles around governance and compliance at an earlier stage will reduce the burden on this when an actual transaction is underway.

A common approach is to disclose issues but to undertake to address them (e.g. by ripping and replacing code that has been wrongly used) before completion of the deal.

Where the target uses Open Source but does not commercially exploit Open Source by distributing products containing Open Source components, the disclosures are likely to be much more wide ranging (and the related warranties less onerous) than where Open Source is core to the business which is the subject of the acquisition or investment.

8.6 Investment in Open Source Businesses

8.6.1 Similarities and differences to M&A process

While in an M&A transaction the owner of the underlying company will change (sometimes along with its senior management), in an investment transaction the existing owners and the management are likely to remain with the business.

For the purposes of this chapter the legal and structural differences between the two types of transaction do not require detailed analysis but a key difference is that there tends to be a less rigorous process of warranty and disclosure and instead there is much more focus on diligence.

8.6.2 The investment lifecycle

Technology businesses typically seek investment as they grow. A startup is likely to have a fast and loose approach to the use of code and a lack of budget to undertake detailed audits and maintain robust risk protocols but as a startup moves to scale up its operations and seek serious investment it will need to start the process of reviewing and managing its Open Source estate.

Here early investors can provide assistance (clearly it is in their interest to get things right early so as to protect their investment when, in due course, further investment comes in) by helping the startup to put in place good practice processes and procedures.

Then as subsequent rounds of investment come into the business, the processes and procedures used to manage its code base can scale up.

8.6.3 Debt and equity

It is worth noting that many investments come as a mix of equity investment (contributing capital to the business in exchange for shares) and debt (lending money to the business possibly with a view to turning that into equity later).

If the business's main asset is its Open Source code base, the investor should think long and hard about the security it has over such an asset. If the Open Source has been published to the world, then should the business become insolvent, taking ownership of the business's assets may be of limited value as its main asset may be freely available for anyone to use.

8.6.4 Risks and controversies

As discussed at section 8.4.2 earlier, where private equity (or other investors seeking a rapid return from an investment in an Open Source-based business) consider making an investment, extreme care needs to be taken to understand the Open Source products, the customer/user base, and the viability of any commercialisation model that it is proposed the business adopt or scale up.

Sometimes, moves to commercialise a well-established (and royalty-free) Open Source product whether by making it 'Open Core' discussed more fully in Chapter 16, something users effectively have to pay to use updated versions of or pay for additional service offerings to make them fully usable, can attract controversy and even backlash from users.

In other instances, the hopes and expectations of management or investors around their ability to commercialise their core Open Source offering prove unfounded as the market (or users) find ways to work around the commercial offering but still access the Open Source code.

It is common for investors to seek assurances from a target's management as to the viability of a business plan so the care here is double edged. Incumbent management/owners and their advisors should consider how strong they really think their future prospects are as, of course, should anyone seeking to invest.

8.7 Insolvency

8.7.1 What happens to Open Source assets on insolvency?

There is a common misconception that Open Source is not owned. This is wrong, someone (or a corporation, a collective, or a foundation) owns any copyright work and Open Source is, at some level, about the licensing of the copyright arising in software code, as is fully explained in Chapters 2 and 3.

As we have seen, there is value in owning Open Source but often the ability to monetise Open Source comes from expertise in how it works and how to improve and enhance it rather than in owning the actual code (which of course has been distributed freely to the world).

In the event an entity that owns a suite of Open Source becomes insolvent, those tasked with seeking to salvage value from the assets of the insolvent business (referred to here as the 'insolvency practitioner') will not be able suddenly to make the codebase proprietary. If the software has been distributed 'to the world' on a free and open basis then that code will remain out there, free and open.

Instead the insolvency practitioner may seek to find value in the knowhow remaining inside the enterprise, but since much of that is likely to be in the minds of the employees who created it there is a risk of no value remaining in an insolvent Open Source business.

If the owner of some Open Source is wound up without any sale of its assets, then while its code may still exist, there will no longer be a legal owner of that code. On a practical level this may not matter as the code will still be used, enhanced, and changed, but legally following a chain of title may become futile.

It is also possible to pick up existing licensed Open Source code from an insolvent company and to fork the project.

8.7.2 ‘Going open’

Where a business has a proprietary code base but is looking at insolvency, the board may decide to make its code base open so that the individuals associated with the company can then move on and use their knowhow to operate the software (which would, by this point, be Open Source).

Such an approach is not without risk and, depending on the exact facts and the insolvency law applicable in the jurisdiction in which the company is based, the insolvency practitioner may be able to set aside any such attempt.

A similar approach can be applied with less risk to code that was already opened by an organisation.

8.8 IPO

8.8.1 Issues with Open Source that arise on a listing

As with M&A and investment, an IPO (i.e. an initial public offering or listing of shares on a stock exchange) sees a company seek to transfer its ownership. The difference with an IPO is that instead of a buyer or trade investor undertaking robust due diligence and agreeing a bespoke M&A or investment agreement, the company and its advisors are under a duty to disclose to the stock exchange (and thus potential buyers of shares) all material facts about the business. Once the IPO takes place, unlike with an M&A deal sale and purchase agreement or an investment deal’s terms, shareholders will have limited contractual rights against the company or those who sold shares when it went public.

Prior to a listing, a company issues a document—or prospectus—describing what it does, how it manages itself, and what it sees as its future prospects.

For some businesses that merely use Open Source, the only reference they may make to Open Source in their prospectus is to state they manage their IT systems in a prudent manner (albeit they will need to ensure they do if they make such a statement), but where a key asset of the business is Open Source much fuller descriptions of the policies relating to the creation and use of Open Source will be required and the business will need to show how it will manage its Open Source assets to deliver value to shareholders.

When preparing a prospectus, the company’s management and its advisors must satisfy themselves that the statements they are making as to the way Open Source is managed and maintained and the potential future value of its Open Source assets (or related revenue streams).

8.8.2 Valuation

Where an IPO is of a well-established Open Source business with strong revenues, it is likely that the prospectus can point to this and make the case that the business is correctly valued. As noted earlier however, where the IPO is seeking investment based on the prospects of commercialisation (or increasing commercialisation) of an Open Source or Open Core product then management of and advisors to the business looking to list will need to be very careful in their assessment of the promises they are making to IPO investors.

8.8.3 Where to list?

Any business contemplating an IPO will engage investment banking-type advisors who will consider which stock exchange has the most advantageous rules for the nature of the activities the business undertakes, investors who understand the industry the business operates in, and that, ultimately, can deliver the best price for the shares.

9
Trademarks

Pamela Chestek

9.1 Introduction

Trademarks play a significant role in the Open Source ecosystem – while copyrights and patents are freely shared in Open Source licensing, trademarks are not. A project's reputation is at stake if it allows malicious software, software of poor quality, or software that does not function as expected, to be distributed under the same name.[1] Open Source projects can, and therefore generally do, exercise their exclusive rights under trademark law.

[1] For example, Elasticsearch filed a trademark infringement lawsuit against Amazon, *Elasticsearch, Inc v Amazon.com, Inc.*, No. 3:19-cv-06158 (N.D. Cal.) (complaint filed 27 September 2019). In it, Elasticsearch alleged that the Amazon products 'Amazon Elasticsearch Service' and 'Open Distro for Elasticsearch', which are based on, but not identical to, Elasticsearch's Open Source products, infringed Elasticsearch's trademark rights. Amazon ultimately renamed its version of Elasticsearch to 'Amazon OpenSearch Service'. 'Amazon Elasticsearch Service is now Amazon OpenSearch Service and Supports OpenSearch 1.0' *AWS News Blog* (September 2021) <https://aws.amazon.com/blogs/aws/ama

Pamela Chestek, *Trademarks* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press.
© Pamela Chestek 2022. DOI: 10.1093/oso/9780198862345.003.0009

The same trademark law applies in the Open Source world as across all other industries. The challenge lies in working in a field that has a practice of freely sharing and owes some of its existence to community contribution and collaborative creation. The trademark practitioner trained in traditional industries may have to rethink their approach to issues. Traditional practices in trademark law are sometimes more maximalist than may be necessary and Open Source trademark counsel should be willing to examine their own beliefs and knowledge of trademark law to ensure that their advice is best suited to the project's ideals.

Some trademark doctrines are also more important in the software field, and Open Source software more specifically, than for other types of goods and services. Trademark fair use, naked licensing, the functionality doctrine, and ownership law are all particularly relevant in the Open Source software field.

Section 9.2 of the chapter covers the basics of trademark law. Section 9.3 provides background on the types of goods and services one typically finds associated with Open Source projects; section 9.4 discusses an aspect of trademark law that is fairly unique to Open Source, the role of community engagement; section 9.5 discusses theories for the lawful use of another's trademark; and section 9.6 closes with a discussion of various ways a trademark owner might try to undo the Open Source licence using trademark law as the vehicle since copyright is largely unavailable.

Because Open Source is international, the chapter attempts to describe trademark law normalised across jurisdictions, largely the US, the UK, and EU. The author is admitted to practise in the US, so the chapter tends towards a US viewpoint. The reader should consider the information here as advisory only and research the specifics of any country's laws before taking action.

9.2 Trademark Law Basics

9.2.1 Definition and function

A trademark is a word, phrase, symbol, design, or characteristic that identifies and distinguishes the source of the goods or services of one party from those of others. Trademarks and service marks can include colours, scents, and sounds.[2] The term

zon-elasticsearch-service-is-now-amazon-opensearch-service-and-supports-opensearch-10/> accessed 14 April 2022. The parties thereafter settled their lawsuit. 'Elastic and Amazon Reach Agreement on Trademark Infringement Lawsuit' *Elastic Blog* (16 February 2022) <https://www.elastic.co/blog/elastic-and-amazon-reach-agreement-on-trademark-infringement-lawsuit> accessed 14 April 2022 ('Now the only Elasticsearch service on AWS and the AWS Marketplace is Elastic Cloud.').

[2] See, e.g., US Reg No 2901090 <http://tsdr.uspto.gov/#caseNumber=76408109&caseType=SERIAL_NO&searchType = statusSearch> accessed 30 November 2019, for 'the color chocolate brown, which is the approximate equivalent of Pantone Matching System 462C, as applied to the entire surface of

'service mark' is used for services and 'trademark' is used for goods, although the words 'trademark' and 'mark' are also frequently used to encompass both goods and services. The terms 'trademark' and 'mark' will be used interchangeably in this chapter for both trademarks and service marks. 'Trade dress' is a term used for the 'get up' or dress of a product, meaning the packaging for the product, the configuration of the product itself, or the décor or environment in which services are performed, such as the interior design of a chain restaurant.

The concept of 'brand' is similar to 'trademark'. A 'brand' is a marketing term used to describe the entire engagement of a consumer with a product or business. The 'trademark' is a vessel for the brand identity, the tangible manifestation that the consumer associates with the product or business.

A trademark is not simply the word or symbol standing alone, but the word or symbol as used in association with particular goods or services. This is what allows the same word 'Delta' to be used as a trademark for an airline,[3] for faucets,[4] and for a dental insurance plan.[5]

A trademark identifies a single, unique source. However, the consumer does not have to be able to identify the source by name. The quality of being a trademark, rather than simply a word, means that consumers recognise that the word or symbol indicates that the goods or services originate from one particular entity, differentiated from others of like kind in the market.

Trademarks are often described as having 'goodwill'. 'Business goodwill' in a larger sense is the value created by the likelihood that a consumer will be a repeat user of the same product or service. This can be because of a favourable or long-term use of a location (such as a gas station on an easily accessible corner which builds repeat loyalty), a product available nowhere else (such as one protected by a patent), or a positive experience with employees. 'Trademark goodwill' is specifically the customer's willingness to acquire (or avoid) a product or service because the customer has a quality association with the trademark, such as wanting to repurchase a product because it has known attributes, to have the cachet of owning

vehicles and uniforms' owned by United Parcel Service of America, Inc.; US Reg No 5467089 <http://tsdr.uspto.gov/#caseNumber=87335817&caseType=SERIAL_NO&searchType = statusSearch> accessed 30 November 2019, for 'a scent of a sweet, slightly musky, vanilla fragrance, with slight overtones of cherry, combined with the smell of a salted, wheat-based dough', i.e. Play-Doh®, owned by Hasbro, Inc.; US Reg No 1395550 <http://tsdr.uspto.gov/#caseNumber=73553567&caseType=SERIAL_NO&searchType=statusSearch> accessed 30 November 2019, for 'a lion roaring' owned by Metro-Goldwyn-Mayer Lion Corp., audio file at <http://tsdr.uspto.gov/documentviewer?caseId=sn73553567&docId=SPE20160602144513#docIndex=11&page=1> accessed 30 November 2019.

[3] US Reg No 924004 <http://tsdr.uspto.gov/#caseNumber=72017621&caseType=SERIAL_NO&searchType=statusSearch> accessed 3 December 2019.

[4] US Reg No 668880 <http://tsdr.uspto.gov/#caseNumber=72030385&caseType=SERIAL_NO&searchType=statusSearch> accessed 3 December 2019.

[5] US Reg No 1665228 <http://tsdr.uspto.gov/#caseNumber=74084185&caseType=SERIAL_NO&searchType=statusSearch> accessed 3 December 2019.

a branded product, or because the consumer sees the mark on a new type of goods and trusts the trademark owner to purvey goods of equal quality in this new field.

Although a trademark always identifies a single source, the way a trademark is used may indicate different types of relationships with that source. A trademark may indicate the source of manufacture, 'I made this'; that the trademark owner controls the quality of the goods and services but did not make them, such as in a franchisor–franchise relationship; that the trademark owner has endorsed or approved the third-party's goods and services, such as licensed sports team apparel and promotional items; or that the trademark owner is simply advertising, such as venue naming rights. The consumer will understand these different relationships based on where the trademark is placed, whether other trademarks are also used, wording that may clarify the relationship, and the consumer's common sense and familiarity with how businesses operate. It will be a trademark infringement to suggest to consumers that any of these relationships exist if they are not true.[6]

These relationships are all commonly found in the Open Source context. The use of a mark as an indicator of 'manufacturing' or source will be use of the mark on the software that is available from the canonical or 'blessed' repository. An example of its use as an indicator of quality might be use by those distributors authorised by the particular project, such as 'Linux' branded distributions; an example of the endorsement relationship might be event sponsorship; and an example of the advertising relationship might be the use of the mark on promotional goods such as stickers and socks.

Where a product is copyrightable content, such as software, the work's title may function only as the name of the work, not as a source identifier, or it may also function as a source identifier. 'Harry Potter' is a trademark for books, movies, and all sorts of related goods, but 'The Sun Also Rises' is not. There is also a policy interest in ensuring that trademark rights do not interfere with the publication of a work by others after the copyright has expired.[7] This interest in reproducing formerly copyrighted works, however, has to be balanced against the interest in protecting the public from confusion, so even where the title owner cannot show the title functions as a trademark, the law may nevertheless prevent deceptive use.[8]

[6] See, e.g., US Trademark Act of 1946, as amended, 15 USC § 1125(a) (2018) (hereafter Lanham Act), stating that it is an infringement for a person to use any word, term, name, symbol, or device (design) where it is likely to cause confusion as to affiliation, connection, association, origin, sponsorship, or approval of goods, services, or commercial activities; Regulation (EU) 2017/1001 of the European Parliament and of the Council of 14 June 2017 on the European Union trade mark (hereafter EU Trademark Regulation) art 9(2)(b) –(c), giving the trademark proprietor the exclusive right to prevent third parties from using a trademark if there exists a likelihood of confusion, including a likelihood of association, or where a sign takes unfair advantage of, or is detrimental to, the distinctive character or the repute of the EU trade mark.

[7] *G & C Merriam Co v Syndicate Pub Co*, 237 US 618, 622 (1915) ('[U]pon the termination of [the copyright on the work] there passes to the public the right to use the generic name by which the publication has been known during the existence of the exclusive right conferred by the copyright.').

[8] Anmerkung zu OLG Düsseldorf, U. v. 24.04.2012-I-20 U 176/11 <http://germanitlaw.com/wp-content/uploads/2012/05/Higher-Regional-Court-Duesseldorf-final.pdf> accessed 17 January 2020

A common misunderstanding is that the name of an Open Source project cannot function as a trademark because the licence for the work permits the creation of a number of different versions of the work, all with different qualities and characteristics, and therefore the trademark function is not satisfied. This is not accurate, as will be discussed in more detail in section 9.3.

9.2.2 Territoriality

Trademarks are territorial. Virtually every country in the world has a trademark registration system and laws regarding the infringement of trademarks. There are some registration systems that cross borders; for example, the single Benelux trademark register is for Belgium, the Netherlands, and Luxembourg. The EU Trademark is a registration enforceable in all the member countries of the EU, overlaying and in addition to each country's national registration system.[9] This is similar to the US, where there are state registration systems as well as a national registration system.

Because each country has its own trademark laws, whether a word or symbol can function as a trademark, whether a trademark can be registered, whether there is a likelihood of confusion between any two given marks, and whether one might have enforceable rights in an unregistered trademark, may be answered differently in each country.

9.2.3 Distinctiveness

The concept of 'distinctiveness' is a term of art that is fundamental to trademarks—the difference between something that is just a word or an attractive design and a trademark is whether the word or symbol is 'distinctive', that is the consumer recognises it as indicating a single source.

The concept is further broken down into 'inherent distinctiveness' and 'acquired distinctiveness', also known as 'secondary meaning'. A word, symbol, or design will be inherently distinctive if, because of its nature, a consumer will immediately

(English translation; holding that defendant could use the 'Enigma' work title for software modified for its hardware platform and configuration as long as the use did not violate generally accepted practices of trademark and commerce, which in the case of Open Source meant that the essential functions of the defendant's version of the software are identical, plug-ins and/or extensions of third parties remain compatible, and the defendant abides by the conditions of the GPL licence).

[9] As a consequence of 'Brexit', trademarks that were registered in the EU as of 31 December 2020 were 'cloned' in the UK trademark registry, so that the owner now has registrations in both jurisdictions. Pending applications were not cloned and the owners have to apply anew to register the trademarks in the UK.

recognise it as a trademark. Where a word, symbol, or design is not inherently distinctive, in some cases it can acquire distinctiveness, meaning that although the consumer might not have immediately grasped that the word or symbol is being used as a trademark, over time through exposure they will have learned to associate the word or symbol with a particular product or service, or family of products or services. A mark that is not distinctive cannot be successfully registered.

Trademark law has developed rules for when a word or symbol will be considered inherently distinctive and when they can never be considered inherently distinctive, or might never acquire distinctiveness. It is conceptualised as a spectrum. On one end, a made-up or 'coined' word will be considered inherently distinctive, immediately entitled to protection as a trademark. Because they have no other meaning, we recognise immediately that they are trademarks. The same is true of 'arbitrary' trademarks, which are dictionary words but used in a way that has no relationship to the product or service, such as CAMEL for cigarettes.

The US has another concept for describing a category of inherent distinctiveness, which is 'suggestive'. A 'suggestive' mark is one that suggests the nature of a product or service or one of its attributes without actually describing the product or service, such as AIRBUS for airplanes.

On the other end of the spectrum, a word that is 'generic', meaning that it is a common name for the product or service, can never function as a trademark, even if consumers associate the word with only one product (such as a category-creating product, like in-line skates). This will happen where the trademark is used as the category name for the goods or services rather than a brand identifier.[10] Trademarks for software in general are susceptible to becoming genericised because any particular software product may have unique attributes not shared by other programs, so consumers use the trademark to refer to the software with those particular characteristics. When there are thereafter new market entrants making similar software with similar attributes, consumers may nevertheless refer to the new entrant's goods using the same term that was meant to be a trademark (see section 9.2.7 for the loss of rights due to genericism).

In addition to generic terms, words, symbols, or designs that are 'functional' will not be protected as a trademark. Functionality is generally at issue when a product's shape is claimed as a trademark, such as uniquely shaped fan blades. If the fan blades are uniquely shaped because they provide a functional benefit, such as moving air more efficiently, then they will not also be protected as a trademark.

[10] An example of a term that a court concluded was a generic term rather than a trademark is '386' for a computer chip. *Intel Corp v Advanced Micro Devices, Inc*, 756 F. Supp. 1292, 1298 (N.D. Cal. 1991). This is also a territorial distinction; 'hoover' is generic in the UK but a registered trademark in the US, US Reg No 5181636 <https://tsdr.uspto.gov/#caseNumber=87172024&caseType=SERIAL_NO&searchType=statusSearch> accessed 27 February 2021. 'Aspirin' is generic in the US but a trademark registered in Germany since 1899, German Reg No 36433 <https://register.dpma.de/DPMAregister/marke/register/36433/DE> accessed 27 February 2021.

Moving back along the spectrum towards inherent distinctiveness, words that are 'merely descriptive'[11] do not immediately function as trademarks but may over time acquire distinctiveness and function as a trademark, and be registrable as such, after they have acquired it. A word will be considered 'merely descriptive' when it states an attribute, feature, end result, or use of the product or service bearing the mark. Examples of merely descriptive terms that acquired distinctiveness are LYFT for computer software for coordinating transportation services[12] and STEELCASE for metal office furniture.[13] Similarly, geographically descriptive terms, surnames, and laudatory terms like 'best' or 'premium' are not inherently distinctive but sometimes can be shown to be functioning as trademarks with appropriate proof.

9.2.4 Registration

In the ideal world, a new name will have been 'cleared' before the project started using it, in other words various relevant sources, including trademark registers, were searched to determine whether anyone else has already registered the same or similar mark for the same or similar services and whether the new user might be inadvertently infringing another's already existing registered or unregistered trademark rights. But probably most Open Source projects will have adopted a name without much thought given to whether others might be using a similar name for a similar product or project. Nevertheless, even though if the project mark is already in use, it may still be useful to have searches performed before registering to learn what obstacles might present themselves during the registration process. Some countries examine trademark applications on 'relative' grounds; that is, whether there is a trademark already registered that the new application will be too similar to. Many trademark owners also have trademark 'watches' which are services that report on the filing or publication[14] of trademark applications similar to their watched mark. The application process itself is therefore likely to bring the

[11] Lanham Act, see note 6, § 1052(e)(1). The equivalent under EU law is trademarks that 'consist exclusively of signs or indications which may serve, in trade, to designate the kind, quality, quantity, intended purpose, value, geographical origin, or the time of production of the goods or of rendering of the service, or other characteristics of the goods or services'. Directive (EU) 2015/2436 of the European Parliament and of the Council of 16 December 2015 to approximate the laws of the Member States relating to trade marks (hereafter Trademark Directive) art 4(c).

[12] US Reg No 4686618 <http://tsdr.uspto.gov/#caseNumber=85743120&caseType=SERIAL_NO&searchType=statusSearch> accessed 1 December 2019.

[13] US Reg No 534526, http://tsdr.uspto.gov/#caseNumber=534526&caseSearchType=US_APPLICATION&caseType=DEFAULT&searchType=statusSearch accessed 12 January 2020.

[14] In virtually every country the trademark registration process includes a step called 'publication', which is publishing the application or registration specifically for the purpose of giving notice to third parties so they have the opportunity to oppose the applications or registrations that they believe are problematic.

project's trademark to the attention of owners of similar trademarks, which should be taken into account before registering.

In addition to relative grounds, trademarks are also examined, and sometimes refused registration, on 'absolute' grounds, which include descriptiveness, genericism, and a number of other statutory bars.[15] A trademark counsel will be able to identify the possible bases for refusal and aid in determining the best procedural approach for registration.

9.2.4.1 Trademarks and service marks

A trademark registration identifies the mark and the goods and services for which registration is granted. The mark can be a word per se, a word in stylised form, a logo, a sound, a colour, trade dress, and so forth. Most countries divide goods and services into 'classes' using a classification system described in an international treaty.[16]

For Open Source, common core classes are 9 (electrical and scientific apparatus goods including software), 38 (telecommunications services), 41 (education services), and 42 (computer and scientific services), but others may also be relevant depending on the function of the software, such as security in Class 45. The relevant classes may change over time as more related goods and services become available. Promotional goods may be included in many classes; for example, Class 16 includes stickers, Class 21 includes glassware, and Class 25 is for apparel.

In almost all countries, one can register a trademark before the use of the mark has started. However, in the US, as a general rule the trademark must be used before the registration will be granted.[17] One can file an application on an 'intent to use' basis to hold a place in line, but the registration will not be granted until the owner has submitted proof of use to the US trademark office. But even in those countries that do not require proof of use before registration, one can cancel a trademark after a statutory period of time, generally three or five years, on the basis that the proprietor is not using the mark.

The legal significance of a registration differs from country to country. The owner of the senior rights will always prevail over the owner of junior rights, but whether or not the mark is registered will also affect who will prevail in a conflict. There are two main legal premises: (i) trademark rights accrue through use, with registration serving as government recognition of the existing rights; and (ii) trademark rights are granted through registration. However, the division described

[15] There are a variety of issues raised around the world, such as deception, geographic misdescriptiveness, that the word is surname, and a prohibition on registering flags and coats of arms.

[16] Nice Convention Concerning the International Classification of Goods and Services For the Purposes of the Registration of Marks, 14 June 1957, as last revised at Geneva, 2 October 1979, 550 UNTS 45.

[17] By international treaty, the foreign owner of a trademark registered in their own country can obtain a registration in the US without having used the mark in the US first, although the owner must still have an intent to use the mark in the US. Lanham Act, see note 6, § 1126(e).

is more simplistic than the laws any country has enacted and a country's trademark system is likely to be a blend of both concepts. For example, in the US an unregistered trademark is as readily enforceable as a registered trademark, except that the owner will have to prove the validity of the mark in court rather than before the US Patent and Trademark Office. In the UK, one can prevent infringement of an unregistered trademark if the wrongdoing rises to the level of 'passing off', which is a false representation that causes confusion or deception and is more difficult to prove than infringement of an unregistered trademark in the US. In the EU, one can rely on prior unregistered rights to oppose or cancel the registration of a junior user but cannot sue for infringement unless the mark is considered a 'well-known' mark.[18] In yet other countries, unregistered rights are of no value at all, unless the mark is well-known. Although it is often not feasible to register every trademark in every country, registration should be the preference rather than hoping that unregistered rights will be adequate.

The Madrid Protocol is an international treaty[19] for the management of International Registrations. Despite the name, an International Registration is not an independently enforceable trademark registration. Instead, the Madrid system is a unified application system. One has a 'basic' application or registration in one's home country. This basic application or registration is used as a basis for filing an International Application, claiming the same mark and description of goods and services as the basic application. At the time the applicant files the International Application, the applicant also requests 'extensions of protection' or 'designates' member countries of its choosing out of the 100+ countries that are members of the Madrid Protocol. Each country thereafter examines the designation under its local legal standards and, if allowed, the designation is enforceable as if it was a registration filed originally in that country.

9.2.4.2 Certification and collective marks

Certification and collective marks are special types of trademarks recognised in some countries. A certification mark is where one certifies the goods or services of a third party. A certification mark cannot be used to certify one's own goods.[20] A certification mark is therefore used for the case where a third party is ensuring that others' goods and services meet a standard, such as the 'Underwriters Laboratory'

[18] 'Well-known' marks are given special treatment by treaty. Paris Convention for the protection of industrial property of March 20, 1883, as revised at Brussels on 14 December 1900, at Washington on 2 June 1911, at The Hague on 6 November 1925, at London on 2 June 1934, at Lisbon on 31 October 1958, and at Stockholm on 14 July 1967, 828 UNTS 305 (hereafter the Paris Convention) art 6*bis*.

[19] Protocol Relating to the Madrid Agreement Concerning the International Registration of Marks, adopted at Madrid, 27 June 1989, as amended on 3 October 2006, and on 12 November 2007 <https://wipolex.wipo.int/en/text/283484> accessed 18 January 2020.

[20] Lanham Act, see note 6, § 1127; EU Trademark Regulation, see note 6, art 83(2).

seal used on electrical equipment.[21] The use restriction mean that a certification mark cannot be used by an Open Source project as a way to provide assurances that third party hosts have authentic software,[22] but a certification mark can be useful to an organisation like the Open Source Hardware Association, which certifies that goods comply with the Open Hardware Definition.[23]

Collective marks are owned by an organisation whose members can use the mark to indicate their membership in the organisation and to identify their own goods and services as coming from a member of the organisation,[24] such as the mark UNICODE used to indicate membership in an association of those who use a computer encoding system utilising 16 bits.[25] An Open Source project could own a collective mark and allow its members to indicate they are members of the project. However, a collective mark would be of low value for uses relating to software goods, since the collective mark indicates only the bona fides of the producer, not that the goods that each produce are uniform.

It is particularly prudent for the Open Source project to register its trademark. Because of the collaborative way that Open Source products are created, there may be many individuals who have a sense of ownership over the software and the project as a whole and there may also be authorised forks or permitted variations of a product. The application process may assist the project in guiding thinking around the question of appropriate ownership and permitted third-party usage.

The freedom for all to make copies of the software and distribute it also lends itself to a claim that the trademark is a generic term that cannot be owned, or that the trademark has lost its source-identifying meaning because it can be used by many (see section 9.2.7). Registration will be a barrier to challenges of this type.

9.2.5 Ownership and licensing

In the US, as a use-based country, the basic premise is that a trademark is owned by the entity that controls the quality of goods or services with which the mark is used. If the application is not filed by the person or entity who actually controls the quality of the goods or services, the application or registration will be invalid.[26]

[21] US Reg No 782589, <http://tsdr.uspto.gov/#caseNumber=72185169&caseType=SERIAL_NO&searchType=statusSearch> accessed 19 January 2020, EU Appln No 017277311, <https://euipo.europa.eu/eSearch/#details/trademarks/017277311> accessed 19 January 2020.

[22] This is simply a licensed use.

[23] US Reg No 5479050 <http://tsdr.uspto.gov/#caseNumber=87473889&caseType=SERIAL_NO&searchType=statusSearch> accessed 19 January 2020.

[24] Lanham Act, see note 6, § 1127; EU Trademark Regulation, see note 6, art 74.

[25] US Reg No 1981995 <http://tsdr.uspto.gov/#caseNumber=74575181&caseType=SERIAL_NO&searchType=statusSearch> accessed 17 January 2020.

[26] For further information on ownership of Open Source trademarks, see Pamela S Chestek, 'Who Owns the Project Name?' (2013) 5(2) *International Free and Open Source Software Law Review* 105.

It can sometimes be difficult to tell who is controlling the quality of the goods and services. Take, for example, a distributor that selects pre-existing products for private labelling. The manufacturer exercises control through its manufacturing process, but the distributor exercises control through its selection of the manufacturer. They may end up in a conflict over who the true mark owner is.[27]

In countries or regions where rights are primarily based on registration, such as the EU, the owner of the mark will be the one to whom the registration was granted, although there may be other defences that invalidate the registration, such as bad faith or fraud. For example, if an opportunist registers a trademark used by another, intending to sue the true owner for infringement or extract payment once the true owner tries to file their own application, the registration may be vulnerable to cancellation for bad faith.[28]

A trademark licence grants permission to use a trademark. They can be expressed in an oral or written agreement or a licence may be implied in conduct. A trademark owner can be both a manufacturer in its own right and licence others to manufacture, for example, to increase capacity. The trademark owner can licence others to create goods or services for convenience, such as outsourcing conference organisation, or because it does not have the facilities or expertise, such as the manufacture of promotional goods.

Use of an Open Source trademark by a third party will be lawful for one of two reasons: the person using the mark is a licensee (see section 9.5.4 explaining that Open Source licences themselves do not grant a trademark licence) or the use is a non-infringing one that trademark law cannot prevent, such as referential use (see section 9.5.3).

It is important for Open Source projects to decide which words, logos, or marks they will register and to establish appropriate guidelines for where use is permissible, particularly where there may be community contribution and use, such as a user group (see section 9.5.6).

9.2.6 Enforcement of trademark rights

Trademark rights are not an absolute right to use a particular word in any context or even within the specific context of the relevant goods and services. One may only prevent another's use of the trademark where the use will cause a very specific type of harm, either a 'likelihood of confusion' or 'dilution'.

[27] It is so common that the Paris Convention states specifically that if a representative or agent registers a proprietor's mark in a different country, the proprietor can oppose or cancel the registration or seek the transfer. Paris Convention (n18) art 6 *septies*.

[28] Several Provisions for Regulating Application for Trade Mark Registration arts 3–4, <http://www.sipo.gov.cn/zfgg/1143015.htm> accessed 18 January 2020.

9.2.6.1 Likelihood of confusion

The fundamental purpose of trademark law is to prevent a 'likelihood of confusion' between the goods or services produced by one entity and those of another, and it will be a trademark infringement where a likelihood of confusion has not been avoided. However, there are as many different legal standards for 'likelihood of confusion' as there are judicial systems.

Nevertheless, the basic inquiry will always consider the degree of similarity of the two trademarks and the degree of similarity of the goods and services. As a rule of thumb, the more similar the marks are the less similar the goods or services will have to be for consumers to be confused, and the more similar the goods and services are the less similar the marks will have to be.

In the US, all the different courts of appeal have different legal tests, but the conclusion is not likely to differ between courts. One typical formulation considers:

- the strength of the mark
- the degree of similarity between the two marks
- the proximity of the products or services
- the likelihood that the prior owner will bridge the gap
- actual confusion
- whether the defendant acted in bad faith
- the quality of the defendant's product
- the degree of care exercised by the consumer in the transaction
- other relevant variables[29]

To elaborate on some of the factors, the 'strength' of the mark refers both to inherent distinctiveness and acquired distinctiveness (see section 9.2.3). The role of strength in a likelihood of confusion analysis is that consumers are less likely to confuse marks that have a significant degree of descriptiveness or ubiquity, even if the mark has acquired distinctiveness and is therefore functioning as a mark. For example, the word 'Enterprise' is commonly used for a version of a software program optimised for a large-scale corporate user. Because of consumers' familiarity with the term and its common use, they have acclimated to seeing the same word used by different companies and therefore are less likely to believe that two software programs sharing only the common word 'enterprise' are from the same source.[30]

'Proximity of the goods' means how similar the parties' goods and services are to each other, and 'bridging the gap' means the likelihood that the two parties'

[29] *Polaroid Corp v Polarad Elecs Corp*, 287 F.2d 492, 495 (2d Cir. 1961).

[30] For example, MYSQL ENTERPRISE (EU Reg No 005708532), PUPPET ENTERPRISE (EU Reg No 013258901), and HEWLETT PACKARD ENTERPRISE (EU Reg No 013906185) are all registered in the EU for goods or services related to databases.

goods and services might move closer together. Lumber and storage sheds are not particularly similar, but a lumberyard might 'bridge the gap' by selling premanufactured sheds.

Legal intervention is available upon proof of a likelihood of confusion in order to prevent harm to consumers. Actual confusion, such as misdirected complaints or goods returned to the wrong party, is compelling evidence that the harm is indeed occurring. Actual confusion is therefore evidence that the legal wrong, likelihood of confusion, is occurring.

Bad faith is considered on the theory that if one is trying to create confusion, one is likely to succeed in doing so. Thus good faith is not generally relevant, only bad faith.

Some transactions are done hastily, such as the purchase of a small food item. Consumers spend a great deal of time investigating other transactions, such as purchasing a vehicle. The less care spent on the transaction, the more likely it is a consumer will be confused.

In the EU, trademark infringement is either per se, without requiring further proof of likelihood of confusion—in the case of 'double identity', meaning that the marks are the same and the goods and services are the same[31]—or upon proof of likelihood of confusion, where the accused trademark is identical with, or similar to, the EU trade mark and is used in relation to goods or services which are identical with, or similar to, the goods or services for which the EU trade mark is registered.[32]

However, trademark infringement is not simply confusion as to origin of a product or service. As discussed in section 9.2.1, use of a trademark may indicate different kinds of relationships, such as approval or endorsement. Confusion about any of these types of relationships will also be a trademark infringement.[33]

9.2.6.2 Dilution

Trademark 'dilution' refers to use of a mark by a third party in a way that will tend to weaken the uniqueness of the owner's mark. A trademark is associated with goods and services, allowing coexisting uses where the goods and services are sufficiently distant that consumers will not be confused (see section 9.2.1, giving examples of coexisting use of the same words for different goods and services). However, there is a theory that a third-party's use of a mark strongly associated with a different entity can nevertheless harm the original user even in the absence of confusion. One harm is in the form of 'blurring', where, instead of one unique source association,

[31] EU Trademark Regulation, see note 6, art 9(2)(b).

[32] EU Trademark Regulation, see note 6, art 9(2)(c).

[33] Lanham Act, see note 6, § 1125(a) (including confusion as to the affiliation, connection, or association of one person with another, or confusion as to the origin, sponsorship, or approval of goods, services, or commercial activities); EU Trademark Regulation, see note 6, art 9(2)(c) (stating that likelihood of confusion includes the likelihood of association).

the consumer now makes two, albeit non-confusing, associations. An example would be GOOGLE used for sunglasses—the consumer is aware that Google does not manufacture sunglasses, but now associates the word 'Google' with two things, search engines and sunglasses.

Another type of harm is 'tarnishment', which is where the junior user's product is unsavoury, unflattering, or offensive and that negative association is visited on the senior user also, to its detriment. Another category sometimes categorised as dilution is 'free riding', where another takes unfair advantage of a consumer's positive association with a trademark for their own gain, such as using too much of a mark in comparative advertising.[34]

9.2.7 Abandonment and genericism

There are a number of ways that trademarks can be invalidated, but two are most relevant in the field of Open Source software: abandonment, more particularly a type of involuntary abandonment called naked licensing; and loss of trademark significance, more particularly where a once distinctive term becomes the generic word for the goods or services.

9.2.7.1 Naked licensing

'Naked licensing', a concept fairly specific to US law, is where the licensor makes no effort to control the quality of the goods or services of its licensee. Some courts take a harsh view of the practice, punishing a licensor because of the possibility that uncontrolled licensees will produce goods of varying quality, in theory harming consumers. In its most harsh implementation, no proof that the goods have varied or that there was any harm to a consumer is required.[35] The principle has been applied even in cases where there is only one uncontrolled licensee.[36] Other courts apply a less draconian standard, requiring that the uncontrolled licensing have actually resulted in a loss of trademark significance before a mark is forfeited through naked licensing.[37]

The risk of a naked licence challenge is of concern to Open Source projects because the Open Source licence allows anyone not only to reproduce the software but also modify it. Different distributors' versions may differ from the canonical source, including in potentially significant ways. If the Open Source project allows

[34] *L'Oréal SA & Ots v Bellure NV & Ots* [2010] ECJ C-487/07.

[35] *Barcamerica Int'l USA Tr v Tyfield Importers, Inc*, 289 F.3d 589, 598 (9th Cir. 2002) ('The point is that customers are entitled to assume that the nature and quality of goods and services sold under the mark at all licensed outlets will be consistent and predictable.').

[36] *Eva's Bridal Ltd v Halanick Enter, Inc*, 639 F.3d 788 (7th Cir. 2011).

[37] *Kentucky Fried Chicken Corp v Diversified Packaging Corp*, 549 F.2d 368, 387 (5th Cir. 1977).

any and all modifications of the software to be branded with the project trademark, it risks a successful challenge that the trademark is invalid due to naked licensing (see section 9.5.6 discussing including licences in trademark guidelines). There is a trial court decision rejecting a claim of naked licensing premised solely on the theory that software was available under a General Public License (GPL) and a GNU Affero General Public License (AGPL),[38] but an Open Source trademark owner must still ensure that it does not grant trademark licences for unfettered use and adequately exercises control over the uses pursuant to the licences it does grant.

9.2.7.2 Loss of trademark significance through genericism

A trademark will not function as a trademark if instead the word or design is 'generic'. A generic term is one that that identifies products and services generally, not specific to any particular source. A term originally coined as a trademark can evolve into the generic term for the category, such as 'escalator' and 'trampoline', both of which were registered trademarks in the US but later invalidated because they had become generic terms.[39]

Because it is a question of consumer perception, a trademark owner may not be able to prevent genericide of its own mark. Consumers may assign meaning no matter what kind of effort the trademark owner employs to prevent it. However, a trademark owner that actively encourages generic use is likely to succeed in losing its trademark rights. While all trademark owners want their brand to 'own the category', encouraging consumers to treat the brand as synonymous with the category will teach consumers that the term is the generic term for the entire category, not an indicator for one particular source for the type of good.

Software in general may be at a higher risk than other kinds of goods for genericide because software products tend to have unique characteristics or functionality, so there may not be an apt or known common name that adequately describes the software. If the trademark owner is not on guard, the trademark is then used as that category name for all new entrants into the field. The trademark owner therefore needs to ensure that it is doing what it can to differentiate the mark from the category for the consumer's benefit. Xerox Corporation periodically runs advertising campaigns reminding consumers that the correct term for reproducing document is 'photocopying', not 'Xeroxing' and Velcro BVBA runs the same type of campaign for 'hook and loop fastener'.

[38] After stating that the GPL and AGPL licences do not incorporate a trademark licence, concluding: 'Defendants have not identified any case, and the Court is not aware of any, in which a trademark owner was found to have engaged in naked licensing where no trademark license existed.' *Neo4j, Inc v PureThink, LLC*, 480 F.Supp.3d 1071, 1078 (N.D. Cal. 2020).

[39] *Haughton Elevator Co v Seeberger*, 85 USPQ 80 (Comm'r Pat 1950); *Nissen Trampoline Co v Am Trampoline Co*, 193 F. Supp. 745, 129 USPQ 210 (S.D. Iowa 1961).

9.3 Open Source Projects, Products, and Services

Open Source projects produce a product, software, but Open Source projects will have goods and services that extend far beyond that. Open Source projects typically create documentation and often provide support services through support-dedicated live chat channels. Projects may provide training, such as local meetups or conferences. They will often create, or allow the creation of, promotional goods like apparel, stickers, mugs, and pens. The Open Source project will be able to register, and assert rights in, its trademark for the software itself and for these additional goods and services.

Beyond that, and perhaps uniquely different from other kinds of businesses, the Open Source project is also a community of contributors. The Open Source project often consists of a group of individuals who are working in a communal way towards a common goal. In addition to producing software, the project may do work dedicated to the enrichment and development of the interests of the community or the improvement of society. It may be a public interest charity performing charitable services, like fundraising to provide financial support to underprivileged members.

The Open Source project will be the producer of the goods and services itself, such as software, documentation, and support services, but it is also likely to be the licensor of the trademark. Typical licensing relationships in Open Source are permissions given by the Open Source project to fans, local groups, or sponsors to create promotional goods or for the outsourcing of event organising. A licence might be granted to one who is going to represent the software project at a booth at a conference, allowing a person to indicate they represent the software project when they have no official relationship with the project. As the trademark owner, the Open Source project has the right to determine in exactly what ways others may use its trademark and in what ways they cannot. However, because of the potential loss of rights through naked licensing (see section 9.2.7.1 on naked licensing), the Open Source project is well-advised to ensure that the scope and requirements of the licences are clear and met by its licensees (see section 9.5.6 on using trademark guidelines to grant trademark licences). At the same time Open Source projects must balance giving their communities enough rights to maintain engagement and recognise the contribution from those communities.

9.3.1 Licensed redistribution

The Open Source licence permits the reproduction and modification of the software by third parties. For purposes of legal analysis, this can be analogised to a trademark licensor engaging another to manufacture goods on its behalf.

Where the trademark owner enlists others to manufacture or offer services, the actual manufacturer or service provider is a trademark licensee and the trademark owner-licensor will dictate to a greater or lesser degree what characteristics the goods or services will have; for example, materials, dimensions, tolerances, manufacturing processes, and quality control checks (if the licensor does not dictate to some degree the quality of the goods, or at least inspect them, the licence is a naked licence as discussed in section 9.2.7.1). If the manufacturer-licensee meets the standards set by the trademark owner-licensor, the manufacturer will be allowed to use the trademark on the goods. If the goods are substandard but the manufacturer nevertheless uses the mark on the goods, the goods will be infringing (see section 9.5.6 for discussion of standards that an Open Source project will want to consider in developing its trademark licence).

9.3.2 Distribution of unmodified software by others without a trademark licence

Trademark exhaustion or exhaustion of rights, also known as the 'first sale' doctrine, allows a third party to use a trademark to resell a product, but this doctrine extends only to stocking, displaying, and reselling an existing, tangible product.[40] In the EU, this may mean that where goods or services are made available in a member state, that exhausts rights across all states under the concept of free movement of goods and services upon which the EU is based.

As applied to Open Source, while there is a copyright licence that allows for the lawful creation of copies, if exhaustion is to apply to the trademark use, one may use the trademark for only those copies distributed in the exact form of the software as provided by the project owner; that is, for executable code only in executable form and for source code only in source code form.[41]

9.3.3 Distribution of modified software without a trademark licence

Although the Open Source licence allows anyone to reproduce the software, those reproducing it are not entitled to represent that modified software is the same as the original, as using the same trademark for the modified software would do. The

[40] *Sebastian Int'l Inc v Longs Drugs Stores Corp*, 53 F.3d 1073, 1076 (9th Cir. 1995); *Beltronics USA Inc v Midwest Inventory Distrib*, LLC, 562 F.3d 1067, 1072 (10th Cir. 2009) (hereafter Beltronics); Trademark Directive, see note 12, art 15.

[41] The Slackware project is an example of this standard. Slackware, 'Slackware Trademark Policy' <http://www.slackware.com/trademark/trademark.php> accessed 16 December 2019 ('In order to be called "Slackware", the distribution may not be altered from the way it appears on the central FTP site').

exhaustion doctrine also does not extend to use of the mark for a different product altogether; for example, executable code created by a third party from project source code.

Where the product, in this case code, is not identical to what has been distributed by the trademark owner, the question of whether it can be distributed under the same mark in the absence of a licence then becomes whether there are material differences between the original goods and what the defendant is distributing.[42] If there is no material difference, the product will not be considered infringing. The situation has not been addressed in the unique context of Open Source, but materiality does often come up in two situations: parallel imports, also known as 'grey market' goods, and cases where goods have been repaired or remanufactured.

Typically, parallel imports are thought of as foreign-manufactured goods sold in a different country without the consent of the trademark holder, but the legal theory also applies to domestic goods. There will be material differences, and therefore a trademark infringement, when the trademark owner's quality control measures have been thwarted, such as by removing a manufacturer's Unique Production Code (UPC) when the manufacturer used it for quality control purposes.[43] Applying this to the Open Source context, if a digital signature is used for the quality control of Open Source software product, much like a UPC, distributing code without the correct signature is likely to be considered a material alteration.[44] Thus, an Open Source project is well within its rights to prevent use of the trademark for anything but its own signed files but may elect to tolerate some changes. The Mozilla trademark policy reflects the former position[45] and the Document Foundation the latter.[46] It will be helpful to users for the Open Source project to state its position publicly in its trademark guidelines.

[42] Beltronics, see note 40, 1072.

[43] *Zino Davidoff SA v CVS Corp*., 571 F.3d 238, 243 (2d Cir. 2009).

[44] *Zino Davidoff SA v CVS Corp.* (finding that UPC codes on perfume boxes were an adequate quality control measure). Similarly, a product can be repaired or reconditioned and resold under the original trademark, but only so long as the reconditioning or repair is not so extensive that it would be a misnomer to call the article by its original name. *Intel Corp. v Terabyte Int'l, Inc*., 6 F.3d 614, 619 (9th Cir. 1993).

[45] The Mozilla Foundation, 'Distribution Policy for Mozilla Software' <https://www.mozilla.org/en-US/foundation/trademarks/distribution-policy/> accessed 8 December 2019, states 'if you make *any* changes to Firefox or other Mozilla software, you may not redistribute that product using *any* Mozilla trademark without Mozilla's prior written consent and, typically, a distribution agreement with Mozilla'.

[46] See, e.g., The Document Foundation (TDF) trademark policy, which allows others to use the LibreOffice trademark on software in substantially unmodified form where 'substantially unmodified' means 'built from the source code provided by TDF, possibly with minor modifications including but not limited to: the enabling or disabling of certain features by default, translations into other languages, changes required for compatibility with a particular operating system distribution, the inclusion of bug-fix patches, or the bundling of additional fonts, templates, artwork and extensions)'. The Document Foundation, 'Policies & TradeMark Policy' (2017) <https://wiki.documentfoundation.org/TDF/Policies/Trademark_Policy> accessed 17 January 2020.

9.3.4 Ancillary goods and services

An Open Source project may have as wide a range of goods and services as any commercial business. The project may sponsor conferences. The project may create formal training program and a substantial set of documentation. These are all goods and services for which registration can and probably should be sought. Where they are provided by a third party, it is a licensing relationship and the project should undertake appropriate steps to ensure that the third-party provider is a controlled licensee.

Projects will also have promotional goods. These will most likely be licensed goods. For these types of goods, the quality control required may be no more than the selection of an appropriate vendor for the goods.[47]

9.4 The Community Role in Open Source Trademarks

9.4.1 Ownership models

Trademarks must have owners, but ownership in Open Source projects may not be clear because of the various development models (see further Chapter 2). It may be that, when a project starts, one individual is the main decision-maker—they have written the bulk of the code, picked the name, and set up the source code repository and website. In this situation the owner of the trademark would fairly clearly be the individual.

It may be instead that the ownership of the mark vests in more than one person; for example, where two or more individuals collaborate equally to create the project. This, in theory, could be problematic if the two owners acted independently, because it may mean that the trademark is not functioning as a mark, that is as a sole source identifier.[48] Nevertheless, where the individuals are contributing to the same code base the risk is minimal since there is only one product.

Where individuals are acting in concert, they may, in fact, be deemed a common law partnership or unincorporated voluntary association. Neither type of legal

[47] Restatement (Third) of Unfair Competition § 33, cmt. c (1995) ('The expectations of consumers depend in part on the character of the licensee's use. If a licensee uses the trademark of a beer or soft drink manufacturer on clothing or glassware, for example, prospective purchasers may be unlikely to assume that the owner of the trademark has more than perfunctory involvement in the production or quality of the licensee's goods even if the manner of use clearly indicates sponsorship by the trademark owner. On the other hand, if the licensee's use is on goods similar or identical to those produced by the trademark owner, purchasers may be likely to assume that the goods are actually manufactured by the owner of the mark. Greater control by the licensor may then be necessary to safeguard the interests of consumers who may purchase the goods on the basis of the licensor's reputation for quality.').

[48] J Thomas McCarthy, *McCarthy on Trademarks and Unfair Competition*, 5th edn (Eagen, MN: Thomson Reuters, March 2021 update) § 16.40 (March 2021 update) (disfavouring joint ownership).

entity requires any filing or formal act to come into existence;[49] instead, they will exist because the law imposes legal structure on concerted acts.

Informal legal organisations are not uncommon. Courts have had to deal with trademark disputes with many kinds of volunteer organisations, like church groups, charities, and clubs. The typical scenario is that a group of individuals will come together to work on a common project or interest, have a falling out, and each then claim to own the name[50]—a scenario that can easily arise with an Open Source project.[51]

With Open Source projects, however, because there generally is some thought about project governance and perhaps documentation of it, the project may be better off than other types of organisations when a court is trying to identify the owner. A 'benevolent dictator' model may mean that the so-called dictator owns the trademark because the person is the ultimate decision-maker about the finished product.[52] A meritocracy model may indicate that it is a partnership or voluntary association that owns the mark.

But there is risk in leaving the question of who owns the mark for a court to sort out. If ownership is challenged in a schism, an adjudicator may indeed find that the project (whether it is an individual, partnership, or unincorporated association) is the owner of the project trademark and prohibit the challenger from using the

[49] See Revised Uniform Partnership Act 1997 (stating that a partnership has been formed where there is 'the association of two or more persons to carry on as co-owners a business for profit forms a partnership, whether or not the persons intend to form a partnership'); *Comm for Idaho's High Desert, Inc v Yost*, 92 F.3d 814, 819–20 (9th Cir. 1996) (noting that under federal law, an 'unincorporated association' is 'a voluntary group of persons, without a charter, formed by mutual consent for the purpose of promoting a common objective'.). It may also be a 'joint venture', *Shain Inv Co v Cohen*, 443 N.E.2d 126, 129 (Mass. App. Ct. 1982) (describing a joint venture as 'a partnership of a sort or, at least, it has many of its characteristics. It differs, however, from a partnership in that it is ordinarily, although not necessarily, limited to a single enterprise, whereas a partnership is usually formed for the transaction of a general business. ').

[50] See, e.g., *Gemmer v Surrey Services for Seniors, Inc.*, No 10–810, 2010 WL 5129241, at *20 (E.D. Pa. 13 December 2010) (senior centre, not the volunteer who thought of the name for and organised a charitable event, owned the trademark for the event); *St. Denis Parish v Van Straten*, Cancellation No 92051378, 2011 WL 5014036, at *4 (TTAB 28 September 2011) (same); *100 Blacks in Law Enforcement Who Care, Inc. v 100 Blacks Who Care, Inc.*, Opposition No 91190175, 2011 WL 1576733, at *4 (TTAB 12 April 2011) (deciding which of two factions of an organisation was the owner of the trademark).

[51] For example, Tim Fox created the Virt.x project while at Vmware. When he departed Vmware for Red Hat, Vmware demanded he turn over the Vert.x Github project, the Vert.x Google Group, the domain vertx.io, and the Vert.x blog. Google Groups, 'An Important Announcement to the Virt.x Community' (2013) <https://groups.google.com/forum/#!msg/vertx/gnpGSxX7PzI/BGhj2PqScY8J> accessed 19 January 2020. Ultimately everyone agreed to move the project to an independent owner, the Eclipse Foundation. Google Groups, 'Community: Please Make Any Objections Known!' (2013) <https://groups.google.com/d/msg/vertx/WIuY5M6RluM/gAvWftxSegUJ> accessed 19 January 2020.

[52] The Linux operating system is an example of a benevolent dictator model: one individual, Linus Torvalds, ultimately decides what is included in the Linux kernel. Linux Kernel Newbies, 'KernelDevViewpoint' (2013) <http://kernelnewbies.org/KernelDevViewpoint> accessed 29 December 2019 (describing how patches ultimately are added to the Linux kernel, with Linus Torvalds deciding what to merge). He also owns the US trademark registration, US Reg No 1916230 <http://tsdr.uspto.gov/#caseNumber=74560867&caseType=SERIAL_NO&searchType=statusSearch> accessed 19 January 2020.

mark. If an Open Source project was unlucky, though, after a falling out it may find that there is a stalemate and no-one will be allowed to use the name from then onwards.[53]

It is therefore best to remove as much ambiguity as possible about who owns the trademark. In practical terms, this means that the project should publicly state who owns the mark, make it clear who may act on behalf of the trademark owner, and allow only the owner to enter into agreements regarding the marks. For example, trademark guidelines should name the owner and provide contact information for how to reach someone with authority to permit use of the mark.[54]

9.4.2 Enforcement

The legal bases for enforcement of trademark rights are the same for Open Source marks as they are for any other trademarks. What is different is the social environment in which trademark enforcement takes place. Because the Open Source ethos is one of sharing, the accused party may believe that their use of the name is either authorised by the software licence or should be tolerated in the spirit of sharing. Some Open Source participants take a position that no intellectual property rights should be enforced, including trademark rights. A community member may contact an infringer directly on behalf of the project, an action that is consistent with the project's practice of openness and transparency, but that may put the legal case in a weaker position.

The communal nature of the field and the multiple stakeholders in a project may also create some unexpected effects in conflicts with non-Open Source parties. When an Open Source trademark is attacked, the project community, as well as the Open Source community at large, is likely to rise to defend the trademark.[55] The Open Source project also may not be willing to agree with terms typically found in settlement agreements, such as confidentiality of payments or agreeing not to speak publicly about the case.

[53] See, e.g., *LunaTrex, LLC v Cafasso*, 674 F. Supp. 2d 1060, 1062 (S.D. Ind. 2009); *Liebowitz v Elsevier Sci Ltd*, 927 F. Supp. 688, 696 (SDNY 1996).

[54] See, e.g., Gnome Foundation, 'Legal and Trademarks' <https://www.gnome.org/foundation/legal-and-trademarks/> accessed 29 December 2019 (stating: 'One of the functions that the GNOME Foundation provides is to act as the legal owner for such GNOME project assets as the GNOME name and the GNOME foot. We must protect these trademarks in order to keep them. Therefore, we have some guidelines for their use and a standard agreement for user groups. These cover many common situations; if you need permission to use the GNOME trademarks in other ways or have other questions, please contact licensing@gnome.org.').

[55] The GNOME Foundation was able to raise over US$100,000 when Groupon tried to adopt GNOME as a trademark for a point-of-sale system, after which Groupon abandoned its trademark applications. Wayback Machine capture of GNOME Foundation, 'Thank you for helping the GNOME Foundation defend the GNOME trademark!' <https://web.archive.org/web/20141114123747/http://gnome.org:80/groupon/> accessed 20 December 2019.

9.5 Lawful Use of Others' Trademarks

Absent a contract, a trademark owner cannot stop lawful uses of its trademarks by third parties. This section describes the ways that a third party may lawfully use a project's trademark. There are, however, no bright lines. Legal issues in trademark law are generally fact-intensive, with a minor change in the factual situation making the difference between an infringing use and a non-infringing use and the ultimate legal conclusion in the hands of the courts.

9.5.1 Non-confusing use

Simplistically, a non-confusing, non-diluting use is not actionable. Section 9.2.6.1 describes how trademark infringement takes into account the similarity of the marks, the similarity of the goods and services, the parties' trade channels, and the strength of the trademarks, and other factors. Where the marks are sufficiently different, or the goods and services are sufficiently different, the relevant audience will not make an association between the two and no legal wrong has occurred. This is the reason that FORD can be used by both a car company[56] and a modelling agency.[57]

9.5.2 'Forking'

The term 'forking' has two meanings. The term 'fork' is sometimes used to mean a branch created in a version control system. However, the term 'fork' was originally used to describe when a developer would elect to exercise their rights under the licence to copy and use the software but wanted to take the software in a different direction. The developer therefore created a new project that started with the same code but thereafter diverged. LibreOffice is a fork of OpenOffice, MariaDB a fork of MySQL, Jenkins a fork of Hudson, and EGCS a fork of GCC, which later was renamed back to GCC.

When the software diverges, it will be confusing for the two projects to share the same name, or names that are highly similar. It also may be that the original project is unhappy with the fork and is not willing to entertain the use of a mark too similar to the original.[58] Whether there is likelihood of confusion will be assessed in the

[56] FORD, EU Reg No 004670618 <https://euipo.europa.eu/eSearch/#details/trademarks/004670618> accessed 19 January 2020.

[57] FORD MODELS, EU Reg No 005188412 https://euipo.europa.eu/eSearch/#details/trademarks/005188412 accessed 19 January 2020.

[58] See, e.g., Hudson Labs, 'Hudson's Future' (2011) https://web.archive.org/web/20110112133740/http://www.hudson-labs.org/content/hudsons-future> accessed 19 January 2020 (because the current

same way that it will be for goods not sharing their origin, with the additional fact that the similarity of the goods is a given.

The second kind of fork is where the name of the project will also be copied over when a branch is created in a version control system. Arguably, the project making the code available implied a trademark licence by using a version control system that will force, at least initially, the use of the same name. This use may be justified as a referential use, as discussed in section 9.5.3. Nevertheless, once the software is modified this situation will most likely be analysed in the same way discussed in section 9.3.3.

9.5.3 Referential use

The need to use another's mark occurs frequently in the software industry as a whole because it is often necessary to include information about the compatibility of software. Users must be told that the application software is compatible with specific operating systems or what dependencies are required.

The doctrine of referential or nominative fair use allows a defendant to use a plaintiff's trademark to identify the plaintiff's goods as long as there is no likelihood of confusion about the source of the defendant's product or the mark-holder's relationship with the defendant.[59] For the defence to apply, the trademark use must be in reference to the original product, not the copyist's.[60] The requirement that likelihood of confusion will not occur will generally mean that the referred-to mark is not used more than absolutely necessary to convey the needed information. Under this standard, it will rarely be the case that the use of a logo can be justified as required for the purpose of conveying the requisite information; generally the word alone will do.

In the US, the situation is addressed by interpreting trademark law doctrine. The EU approaches this situation through the Directive concerning misleading and comparative advertising.[61] The Directive lists a number of conditions that must be met for another's mark to be used in advertising, including that the use not create confusion, that it not take unfair advantage of the referred-to mark, and that it objectively compares the goods or services.[62]

owner would not agree to a transfer of the name 'Hudson', the fork chose the name 'Jenkins' which 'evokes the same sort of English butler feel'.).

[59] *Tiffany (NJ) Inc v eBay Inc*, 600 F.3d 93, 102 (2d Cir. 2010), quoting *Merck & Co v Mediplan Health Consulting, Inc*, 425 F. Supp. 2d 402, 413 (SDNY 2006).

[60] *New Kids on the Block v News Am Publ'g, Inc*, 971 F.2d 302, 308 (9th Cir.1992); *Century 21 Real Estate Corp v Lendingtree, Inc*, 425 F.3d 211, 214 (3d Cir. 2005).

[61] Directive 2006/114/EC of the European Parliament and of the Council of 12 December 2006 concerning misleading and comparative advertising (2006) (hereafter Advertising Directive).

[62] Advertising Directive, see note 61, art 4.

However, a person who offers modified software, or builds a new executable file from source code and labels it with the trademark is not using that trademark nominatively to describe the original project's executable code goods. Rather, it is using the mark as the name for its own newly created version of the product.[63] The use for new goods that are the developer's own creation will not be a referential, nominative fair use.[64]

9.5.4 Trademark licences in Open Source licences

As a general rule, the Open Source licence does not include, either expressly or impliedly, a trademark licence.[65] The author is aware of only one Open Source Initiative-approved licence that imposes a duty to use a trademark.[66] To the contrary, a number of licences state expressly that no trademark licence is granted in the Open Source licence.[67]

The Apache License version 2.0[68] is sometimes described as having a trademark licence. It refers to trademarks in two places: 'You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work' and 'This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE

[63] See, e.g., OpenJDK, 'OpenJDK Trademark Notice Version 1.1' (2008) <http://openjdk.java.net/legal/openjdk-trademark-notice.html> ('The Name may also be used in connection with descriptions of the Software that constitute "fair use," such as "derived from the OpenJDK code base" or "based on the OpenJDK source code.').

[64] See *Neo4j, Inc. v PureThink, LLC*, No. 5:18-CV-07182-EJD, 2021 WL 2483778 (N.D. Cal. 18 May 2021) (holding that modifying Open Source Neo4j software, but calling the modified version 'Government Packages for Neo4j' and 'Neo4j Enterprise' was not a nominative fair use but an infringing use of the trademark for the defendant's own product).

[65] For a discussion about why the Open Source Definition and the Free Software Definition do not address trademark rights, see Pamela S Chestek, 'The Uneasy Role of Trade Marks in Free and Open Source Software: You Can Share My Code, But You Can't Share My Brand' (2012) 7 *Journal of Intellectual Property Law and Practice* 126, 129–30. There is also a US decision stating expressly '[t]he GPL and AGPL are copyright licenses, not trademark licenses'. *Neo4j, Inc v PureThink, LLC*, 480 F.Supp.3d 1071, 1077 (N.D. Cal. Aug. 20, 2020).

[66] 'Common Public Attribution License Version 1.0' <https://spdx.org/licenses/CPAL-1.0.html> accessed 19 January 2020. The author could not find any software licensed under this licence. In the Debian Project's opinion, the licence is not an Open Source licence because of the attribution requirement. Debian Project, 'DFSGLicenses' <https://wiki.debian.org/DFSGLicenses#Licenses_that_are_DFSG-incompatible> accessed 8 December 2019.

[67] See, e.g., 'Academic Free License version 3.0' § 4 <https://spdx.org/licenses/AFL-3.0.html> accessed 19 January 2020; 'Attribution Assurance License' § 3 <https://spdx.org/licenses/AAL.html> accessed 19 January 2020; 'Microsoft Public License' § 3(A) <https://spdx.org/licenses/archive/archived_ll_v2.4/MS-PL.html> accessed 19 January 2020; 'Mozilla Public License version 2.0' § 2.1(a) <https://www.mozilla.org/en-US/MPL/2.0/> accessed 19 January 2020.

[68] The Apache Software Foundation, 'Apache License, Version 2.0' <https://www.apache.org/>licenses/LICENSE-2.0> accessed 18 January 2020.

file.' The first sentence is a requirement that one keep trademark notices but does not require that a trademark be used as the product name. The latter sentence is best understood as an acknowledgment that others may use the Apache trademark referentially (see section 9.5.3), a right that is implicit in all the other Open Source licences.

At one point the Mozilla project used a technical approach to prevent the use of its mark for modified software. It had a 'branding switch', asking that those who modified the software build a version using a switch that would remove the official branding.[69] As another example, the Fedora Project offers a set of 'generic logos' that can be used to replace the Fedora trademarks without breaking any of the functionality of the software if a user is not willing to comply with the Fedora trademark restrictions.[70]

Some Open Source projects, as advocates of free culture, have granted copyright licences for their logos.[71] This, however, is not a grant of a trademark licence: although another may use the design as a graphic design, the project can still enforce its trademark rights against a use of the logo for similar goods and services. There is some risk, however, that a court might decide that the express copyright licence can be construed as granting a licence for all uses, including trademark-infringing ones. Projects may want to consider whether the policy choice of granting a copyright licence for a trademark is worth the risk of third-party use of their mark in a way that might create confusion.

9.5.5 Trade dress

In the context of software, 'trade dress', also called 'get up', will be the appearance or 'look and feel' of a graphical interface for the product. When copying Open Source that has graphical elements, the reuse of the software will necessarily reproduce the look and feel of the origin software. If, however, a project wants to retain exclusivity of its look and feel, as with trademarks it may have to contend with defences that the trade dress was impliedly licensed or that the trade dress is functional (see section 9.6.1) and therefore the downstream user's use is lawful. A project can likely avoid the problem altogether by segregating its trade dress into easily removed folders[72] or not making the design elements available in publicly accessible source code.

[69] Debian Project, 'Debian Bug report logs—#354622, Uses Mozilla Firefox trademark without permission' (2006) <https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=354622;msg=20> accessed 19 January 2020.

[70] Fedora Project, 'generic-logos' <https://pagure.io/generic-logos/tree/master> accessed 19 January 2020. The generic logos are licensed under GPL and LGPL licences.

[71] See Debian Project, 'Debian Logos' <https://www.debian.org/logos/> accessed 8 December 2019 (granting a licence to the official logo under the GNU GPL version 3 or alternatively the Creative Commons Attribution ShareAlike 3.0 Unported License).

[72] See note 80.

9.5.6 Trademark guidelines and policies

It is common in the software industry for software companies to publish guidelines on how others may use their trademark. For proprietary companies, trademark guidelines are often just instructions on how the trademark owner would like third parties to use their mark, with guidance on matters such as where to place the ® symbol and not to use the mark a noun. However, because Open Source software includes the right to reproduce the software, and because Open Source software projects will often want to allow third parties at least some use the mark, trademark guidelines for Open Source projects become more critical.

The Open Source project trademark guidelines will provide information to the user about what the project considers a lawful use of the mark, in particular guidance to redistributors and modifiers about when the mark may be used for reproduced software and when it may not be (see section 9.3.3 on use of the mark on modified software). Although it may ultimately be a legal question, downstream users of the software will know the project's view and that they may use the trademark in ways described by the trademark owner without any risk.

Unlike proprietary software trademark guidelines, the Open Source trademark guidelines may also grant licences (see section 9.3.1 on licensing Open Source trademarks). For example, the ANDROID mark can only be used for 'Android-compatible' devices,[73] which are devices that meet Google's well-defined, testable standard for compatibility.[74] If the trademark guidelines provide a sufficiently detailed description of the quality of the goods and services with which the mark can be used, a naked licensing defence may be avoided (see section 9.2.7.1). For example, standards for promotional goods might consist of approved vendors or manufacturing standards for the goods, for example '100% heavyweight cotton'. A licence allowing a local meet-up group to use the trademark might include parameters on cost to attend and limits on the subject matter of the meetings.[75]

9.6 Attempts to Limit Competition with Trademarks

Some have tried to find a way to use trademarks to monetise Open Source directly or indirectly when the most common mechanism for revenue generation, the grant of a copyright licence in exchange for payment, is unavailable. They may consider

[73] Android Developers, 'Brand Guidelines' <https://developer.android.com/distribute/marketing-tools/brand-guidelines> accessed 29 December 2019 (stating that the 'Android' name and the Android logo are not part of the assets available through the Android Open Source Project).

[74] Android Source, 'Android 10 Compatibility Definition' <https://source.android.com/compatibility/android-cdd.html> accessed 20 December 2019.

[75] Model Trademark Guidelines is a set of model guidelines designed for Open Source projects. 'Model Trademark Guidelines' <http://modeltrademarkguidelines.org/index.php/Home:_Model_Trademark_Guidelines> accessed 15 January 2020.

instead the licensing of another proprietary right, the trademark, for a revenue stream or they may want to force the use of the project trademark for purposes of advertising. The strategies described in the following subsections are not likely to succeed.

9.6.1 Functional use of trademarks

One concept is that placing the trademark in the source code, such that the software will be non-functional if the trademark is removed, and then selling a licence to the trademark, can be used to generate revenue. The trademark may be used in file names or commands, or it can be an image file that breaks the build if not present.

These efforts are likely to fail under the trademark functionality doctrine. One cannot infringe a trademark if the trademark is 'functional'. A product feature is functional if it is essential to the use or purpose of the article or if it affects the cost or quality of the article.[76] In *Sega Enterprises Ltd. v Accolade, Inc.*,[77] Accolade produced game cartridges that were compatible with the Sega gaming console. Game cartridges produced by Sega used what Sega called its 'trademark security system'. A game cartridge contained an initialisation code, four bytes of data consisting of the letters 'S-E-G-A', that prompted a screen display of the Sega trademark. Sega testified that it had used the trademark this way deliberately so that Sega would have a claim for trademark infringement against counterfeiters. Accolade reproduced the initialisation code so that its cartridges would play on the Sega console, prompting a display of the trademark even though the cartridge was not an authentic Sega cartridge. When Sega sued Accolade for both copyright and trademark infringement, the court held that Sega had used its trademark in a functional way and Accolade's use of the initialisation code was not a trademark infringement.[78]

One might also run into GPL compliance problems if trademarks are used in a way that interferes with the operation of the software. According to Richard Stallman,[79] if it is easy to find and remove the trademarks, restrictions on the re-use

[76] *Inwood Labs, Inc v Ives Labs, Inc*, 456 US 844, 851 (1982).

[77] 977 F.2d 1510, 1531 (9th Cir. 1992), as amended (6 January 1993).

[78] See also *Compaq Computer Corp v Procom Tech, Inc*, 908 F. Supp. 1409, 1423 (S.D. Tex. 1995) (required presence of company name in partition before values would be written was a functional use of a trademark). In a similar vein, Autodesk disavowed any claim that the file extension .dwg could function as a trademark because it was functional, although it could be a trademark when used as a logo. 'Put differently, anyone in the world is free to use ".dwg" as a file extension as far as Autodesk is concerned.' *Autodesk, Inc v Dassault Systemes SolidWorks Corp*, 685 F. Supp. 2d 1001, 1009 (N.D. Cal. 2009).

[79] Richard Stallman is the founder and past president of the Free Software Foundation. Free Software Foundation, 'Richard M. Stallman Resigns' (2019) <https://www.fsf.org/news/richard-m-stallman-resigns> accessed 15 December 2019.

of trademarks is not inconsistent with the GPLv2.[80] Conversely, if it is difficult to remove them, the restrictions on use of trademarks may be considered inconsistent with the GPL family of licences.

9.6.2 Trademarks and 'further restrictions'

Some have suggested that Section 7 of the GNU GPLv3 permits requiring use of a trademark. This is not correct.

In general, one cannot add more restrictions to the GPLv3.[81] Section 7 of GPLv3, however, permits a defined set of supplemental terms, one of which is a term 'Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it'.[82] However, a trademark is neither a 'legal notice' nor 'author attribution'. A trademark is not a legal notice because there is no requirement that a product have a trademark.[83] The 'attribution' to an author is the name of the creator of copyrightable content,[84] not the name of the company vending the goods. The GPLv3 also refers specifically to trademarks in a different subsection of Section 7,[85] demonstrating that the drafters knew the difference between trademarks, legal notices, and attributions.

9.6.3 Requiring display of trademarks

It is not uncommon for an Open Source licensor to believe that it can gain an advantage, at least reputational, by requiring the use of its trademark for its software. There was a point where 'badgeware' licences were popular,[86] with this typical language:

[80] Richard Stallman, '[Savannah-hackers] Re: Issue of Trademark Logo Images in Source Distribution' (2004) <http://lists.gnu.org/archive/html/savannah-hackers/2004-11/msg00508.html> accessed 15 January 2020 ('It is no problem if the program contains trademarked images and names, provided the trademark usage and requirements don't make it difficult in practice to change the program and publish a modified version. In other words, it has to be easy to find and remove the trademarks, if and when the trademark conditions require this.').

[81] GPLv3 § 10.

[82] GPLv3 § 7(b).

[83] In contrast, trademark marking through use of the encircled R symbol or a trademark legend, Lanham Act, see note 6, § 1129, is likely to be considered a trademark notice. However, these are ancillary to trademarks, not the trademark itself. And no marking will be required if the trademark is not present.

[84] *Black's Law Dictionary*, 11th edn (Eagen, MN: Thomason Reuters, 2019) (defining 'attribution right' as 'a person's right to be credited as a work's author, to have one's name appear in connection with a work, or to forbid the use of one's name in connection with a work that the person did not create'.)

[85] GPLv3 § 7(e).

[86] At one point, twenty Open Source software companies were reported to be using this licensing language. Rick Moen, 'When is an Open Source License Open Source?' (2007) <https://web.archive.org/web/20161220023005/http://www.linuxgazette.net/141/misc/lg/when_is_an_open_source_lice

> This License does not grant any rights to use the trademarks 'SugarCRM' and the 'SugarCRM' logos even if such marks are included in the Original Code or Modifications.
>
> However, in addition to the other notice obligations, all copies of the Covered Code in Executable and Source Code form distributed must, as a form of attribution of the original author, include on each user interface screen (i) the 'Powered by SugarCRM' logo . . . .[87]

This licence, and the practice of requiring use of the trademark more generally, is inadvisable for several reasons.

In the licence, 'Covered Code' is defined as both the original code and any modifications of it, where modifications include additions, deletions, and new files.[88] No matter how much the downstream user has changed the code, it will be considered 'Covered Code', with the resulting requirement that the origin code's trademark be displayed. Since the licensor has no control over their modifications, the trademark is at risk of being invalidated as a naked licence (see section 9.2.7.1).

And rather than burnishing the Open Source licensor's reputation, it is equally possible that it will harm the licensor's reputation. Changes made downstream may break the software, or the code may be changed for malicious reasons. For example, the Mozilla Firefox browser, distributed at no cost, was used in a 'subscription trap' scheme to charge for the software.[89] The trademark owner who requires the use

nse_open_source.html> accessed 19 January 2020 ('SugarCRM started the trend, and the other dozen-odd firms (Socialtext, Alfresco, Zimbra, Qlusters, Jitterbit, Scalix, MuleSource, Dimdim, Agnitas AG, Openbravo, Emu Software, Terracotta, Cognizo Technologies, ValueCard, KnowledgeTree, OpenCountry, 1BizCom, MedSphere, vTiger) literally copied their so-called "MPL-style" license, with minor variations.').

[87] 'SugarCRM Public License v1.1.3' <https://spdx.org/licenses/SugarCRM-1.1.3.html> accessed 12 December. The licence continues: 'In addition, the "Powered by SugarCRM" logo must be visible to all users and be located at the very bottom center of each user interface screen. Notwithstanding the above, the dimensions of the 'Powered By SugarCRM' logo must be at least 106 × 23 pixels. When users click on the "Powered by SugarCRM" logo it must direct them back to http://www.sugarforge.org. . . .'

Also, the disclaimer that there is no licence granted to the trademark, yet requiring that the trademark be used, is irreconcilably inconsistent. Frequently software trademark guidelines include a statement that one may use a trademark as long as there is no suggestion of affiliation or endorsement. See, e.g., Wordpress Foundation, 'WordPress Foundation Trademark Policy' <http://wordpressfoundation.org/trademark-policy/> accessed 19 January 2020 ('All other WordPress-related businesses or projects can use the WordPress name and logo to refer to and explain their services, but they cannot use them as part of a product, project, service, domain, or company name and they cannot use them in any way that suggests an affiliation with or endorsement by the WordPress Foundation or the WordPress Open Source project. '). Perhaps a party's claim that no licence is granted while simultaneously requiring use of the trademark is an inartful effort to convey that using the logo is not meant to suggest that there is any affiliation or endorsement by the party of the subsequent distribution.

[88] 'SugarCRM Public License v1.1.3' see note 87.

[89] Michael Kerrisk, LWN.net 'Mozilla's Trademark Enforcement Experience' (2013) <https://lwn.net/Articles/546678/> accessed 19 January 2020.

of its trademark will suffer reputational harm from the substandard software or fraudulent use and may risk creating liability itself for the bad acts of others.[90]

9.7 Conclusion

Trademarks are the only proprietary right that can be fully exercised by Open Source projects. The law treats Open Source trademarks no differently than any other trademark, but the Open Source field has special considerations, both the particular ethos of the parties and the specific doctrines that commonly arise, that the trademark solicitor should consider when engaging in trademark work.

[90] *Kennedy v Guess, Inc*, 806 N.E.2d 776, 786 (Ind. 2004) (holding that trademark licensors may have liability for personal injury claims for products they have licensed).

10
Patents and the Defensive Response

Malcolm Bain and P McCoy Smith

10.1 Patents and Software

As discussed in more detail in Chapter 3, the foundation of Open Source licensing is copyright, and in the beginning, consideration of patent rights and patent licences was not paramount. The BSD license,[1] one of the first Open Source licences created (*circa* 1988), states its licence grant as follows:

> Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
>
> … Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

[1] There are several different variants of the BSD licence; this text is reproduced from the 'BSD 3-clause license'—the most commonly used BSD variant—as found on the Open Source Initiative's website. <https://opensource.org/licenses/BSD-3-Clause> accessed 12 August 2020.

Malcolm Bain and P McCoy Smith, *Patents and the Defensive Response* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Malcolm Bain and P McCoy Smith 2022.
DOI: 10.1093/oso/9780198862345.003.0010

> … Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. …

No express mention is made of patents in this grant, although at least one verb—'use'[2]—that is an exclusive right of a patent holder is recited.[3] Similarly, the MIT License, another early Open Source licence created around the same time as the BSD License, states its grant as follows:

> Permission is hereby granted, free of charge, … to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so

Thus, the MIT License[4] uses at least two of the verbs—'use' and 'sell'—that are exclusive rights of a patent holder. At least one commentator has argued that MIT's open ended grant 'to deal in the Software without restriction', followed by exemplary verbs from copyright and patent rights, confers a complete patent licence.[5]

Nevertheless, concerns have long been raised about the scope of patent rights that might be conferred—or might be withheld—in the early Open Source licences. More recent Open Source licences approved by the Open Source Initiative (OSI)—for example the GNU General Public License version 3 (GPLv3 2007) and the Mozilla Public Licence version 2 (MPLv2 2012)—deal quite extensively with patents. For example, relevant portions of the MPLv2[6] read:

> 2.1. Grants
> Each Contributor hereby grants You a world-wide, RF, non-exclusive license:… under Patent Claims of such Contributor to make, use, sell, offer for sale, have

[2] For example, UK Patents Act 1977 § 60; 35 USC § 271(a).

[3] Despite the fact that the general licence grant of the BSD licence is more than thirty years old, there continues to be a debate as to whether any patent rights are conferred by a licensor that chooses to use that licence with their software. Compare David Kappos and Miling Harrington, 'The Truth About OSS-FRAND: By All Indications, Compatible Models in Standards Settings' (2019) 20(2) *Columbia University Science and Technology Law Review* 240–50 with Van Lindberg, 'OSS and FRAND: Complementary Models for Innovation and Development' (2019) 20(2) *Columbia University Science and Technology Law Review* 251–70.

[4] Open Source Initiative, 'MIT License' <https://opensource.org/licenses/MIT> accessed 18 August 2020.

[5] Scott Peterson, 'Why so little love for the patent grant in the MIT License?' *Opensource.com* (23 March 2018) <https://opensource.com/article/18/3/patent-grant-mit-license> accessed 19 March 2021.

[6] Open Source Initiative, 'Mozilla Public License' <http://opensource.org/licenses/MPL-2.0> accessed 12 August 2020.

> made, import, and otherwise transfer either its Contributions or its Contributor Version.
>
> 2.3 Limitations on Grant Scope
>
> … [N]o patent license is granted by a Contributor … for any code that a Contributor has removed from Covered Software; or for infringements caused by: (i) Your and any other third party's modifications of Covered Software, or (ii) the combination of its Contributions with other software (except as part of its Contributor Version); or under Patent Claims infringed by Covered Software in the absence of its Contributions.
>
> …
>
> 5.2. If You initiate litigation against any entity by asserting a patent infringement claim (excluding declaratory judgment actions, counter-claims, and cross-claims) alleging that a Contributor Version directly or indirectly infringes any patent, then the rights granted to You by any and all Contributors for the Covered Software under Section 2.1 of this License shall terminate.

The GNU General Public License was perhaps the first Open Source License to discuss patent rights in any detail; the second version of the GNU General Public License (version 2, in 1991), indicated that software patents were considered a risk for free software. Version 2 of that license, GPLv2, warned of patent threats in its preamble: '[A]ny free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licences, in effect making the program proprietary….' GPLv2 includes provisions purporting to deal with patents, in a clause referred to by the Free Software Foundation (FSF)—the GPL's authors—as the 'Liberty or Death clause'. '[T]he clause that says if somebody uses a patent or something else to effectively make a program non-free then it cannot be distributed at all.'[7] '[P]atents not only do not assist in the production of innovative software, they can potentially destroy the free software production system, which is the world's most important source of software innovation.'[8] While over the years Open Source licences themselves have become more sophisticated with regard to patents—at least to the extent that they make explicit that those that contribute code under an Open Source licence do not reserve the right to assert their patents against those making use of their contributions—there is only so much licences can do to guard against the threat of patent assertions, as a licence only binds those that make use of the rights granted under that licence. Although the threat of patent assertions made against Open Source by patent

[7] FSFE, 'Transcript of Richard Stallman at the 2nd international GPLv3 conference; 21st April 2006' <fsfe.org/campaigns/gplv3/fisl-rms-transcript.en.html> accessed 12 August 2020.

[8] Eben Moglen, 'Free software matters: Patently controversial' *Moglen Law* (2001) <http://moglen.law.columbia.edu/publications/lu-16.html> accessed 12 August 2020.

holders who are not participants (via contributions, or via exercising licence grants) has been recognised since at least the release of GPLv2 in 1991, it has only been more recently that initiatives involving the Open Source community have been set up to fend off the threat of the use of patents to limit the creation and use of free software. One example is the Open Invention Network,[9] a patent pool for providing patent non-assertion commitments to the GNU/Linux operating system ecosystem.

What seems paradoxical is that patents and free software appear to share the same basic objective: to promote development and innovation through transparency and disclosure. It is on the basis of disclosing and sharing knowledge (in patent applications) or through access to source code (in Open Source) that new inventions or innovations may be made over existing technology, whether in an incremental manner or by 'intuitive' leaps. Even the legal technique established for promoting inventions via the patent system—that is granting exclusive rights that may be exercised by the inventor to control the exploitation of the invention by others—should not have been a problem: a similar legal framework of exclusive rights in the area of copyright has been used by the free software community from the start as the very basis for granting and ensuring software freedoms.[10]

However, there are significant friction areas between the two models or approaches to innovation; particularly the fact that patents provide for exclusive control over all and any implementations of a patented idea—as that idea is defined in a granted patent claim—and not just an expression of that idea as with copyright, which gives rise to problems and potential legal risks for free software. The purpose of this chapter is to explore these issues, to understand how the Open Source community tries to deal with patents with the aim of ensuring software freedoms, and concludes by commenting on proposals that have been made to remedy the situation and mitigate the risks.

Therefore, in this chapter we first look at why patents are relevant to Open Source—briefly, the question of *software patentability* and the differences with copyright, and then, taking into account the free software development and licensing models, we consider what the impacts are for Open Source: the interrelations and frictions areas between free software licensing models and patents. Next, how patents are dealt with by the community from a structural point of view—particularly patent-related licensing provisions in free software licences—is reviewed. A discussion of the litigation environment, specifically as it relates to patent assertions against Open Source, is discussed. Finally, how the risks posed by patents—or the way patents are wielded—to the Open Source community may be mitigated, if not removed entirely, are summarised.

[9] Open Invention Network <http://www.openinventionnetwork.com> accessed 9 March 2021.

[10] Richard Stallman, 'The Free Software Definition' in *Free Software, Free Society: The Selected Essays of Richard M. Stallman*, 2nd edn (Boston, MA: GNU Press, Free Software Foundation, 2002–10) 43–6, available at <http://www.gnu.org/philosophy/fsfs/rms-essays.pdf> accessed 23 August 2020.

10.2 Patents 101: Why Are Patents Relevant to Open Source?

Patents are exclusionary rights[11] granted to inventors over an invention, conveying to the patent holder rights to exclude anyone else from exploiting the invention as claimed in the patent in the specific territory for which the patent is granted, for a limited period. In return, the patent holder is obligated to provide a full disclosure of the invention to the public. Patents are granted on application to territorial patent offices (e.g., the UK Intellectual Property Office), after examination for patentability, as well as other eligibility criteria, under the applicable rules.

10.2.1 In Europe

Within Europe, patents are regulated on a regional basis by the European Patent Convention (EPC), which creates a European patent with potential effects in the territories of the signatories to the Convention, and on national bases by the corresponding national patent laws, for example the UK Patents Act 1977, or the Spanish *Ley 11/1986 de Patentes*. In this chapter we will mainly comment on the EPC provisions with respect to software, though it is important to note that it is the national courts applying the law of the member states who ultimately decide on patent validity or infringement, though they tend to follow the European Patent Office (EPO) practice and Board of Appeal decisions.

The state of patenting for software has long been controversial, and there are many arguments as to whether software does or should constitute patentable subject matter. Patents are granted for inventions in all fields of technology that are new, involve an inventive step,[12] and are capable of industrial application.[13] The EPC does not define what is an 'invention'. It does, however, provide a negative limitation, giving examples of what are not to be regarded as inventions. Relevant for the purposes of Open Source is the specific exclusion, under Article 52(2)(c) EPC, of 'programs for computers'.

However, this exclusion is then limited by Article 52.3, which provides that these items are excluded 'only to the extent to which a European patent application relates to such subject matter or activities as such'. It is these last two words, 'as such', that have caused an ongoing and acrimonious debate about software patentability under the EPC, and also under the European national legislations, many of which provide a translation or approximation of this double exclusion/limitation

[11] Patents are not 'exclusive' rights, i.e. a positive and exclusive right to do something, but rather a negative right to exclude others from implementing the claims granted in the patent document.

[12] In the US, this concept is referred to as 'non-obviousness'. See 35 USC § 103.

[13] EPC, Article 52. In the US, a related—but not completely analogous—requirement is 'usefulness'. See 35 USC § 101.

with regard to software,[14] and which ultimately is the benchmark against which the validity of the European patent is measured.[15]

It is not the purpose of this chapter to review the situation of software patentability within Europe, as we aim to focus on the interaction between software patents—however well or justifiably granted—and Open Source.[16] Suffice to say that the EPO has long been granting patents over what have been named 'computer-implemented inventions' (CII), on the basis that they are granting patents over inventions that have technical character and a technical effect that goes beyond the normal interaction of the software with the computer, although ironically 'technical' is not defined in the EPC.[17] European national courts (with some reticence, it was once thought, in England and Wales, but that has proven not to be so) are upholding those grants.[18] What is more, in the light of the debate about software patentability, the Enlarged Board of the EPO rejected the EPO President's request to undertake a full review of the situation, at the instigation of the English High Court, considering that the 'case law' created by the EPO Boards of Appeal is sufficiently clear.[19]

> Indeed, if the Boards continue to follow the precepts of T 1173/97 *IBM* it follows that a claim to a computer implemented method or a computer program on a computer-readable storage medium will never fall within the exclusion of claimed subject-matter under Articles 52(2) and (3) EPC, just as a claim to a picture on a cup will also never fall under this exclusion. However, this does not mean that the list of subject-matters in Article 52(2) EPC (including in particular 'programs for computers') has no effect on such claims. An elaborate system for

[14] For example, Spanish Patent Act 11/1986, art 4.

[15] The proposed Unified Patent Court, approved by the European Council of Ministers and European Parliament, does not exclude software patents per se, but does have limits to enforcing such patents consistent with European Parliament directives allowing for reverse compilation and interoperability. Agreement on a Unified Patent Court, UPC/en 34 n. 1 (19 February 2013).

[16] There are a significant number of thoughtful papers written on this subject. See Noam Shemtov, 'Software Patents and Open Source Models in Europe: Does the FOSS Community Need to Worry About Current Attitudes at the EPO?' (2010) 2(2) *Journal of Open Learning, Technology & Society (JOLTS)* 151–64; Avi Freeman, 'Patentable Subject Matter: The View From Europe' (2011) 3(1) *Journal of Open Learning, Technology & Society* 59–80; Colleen Chien, 'From Arms Race to Marketplace: The Complex Patent Ecosystem and Its Implications for the Patent System' (2010) 62 *Hastings Law Journal* 297–356; Mark Lemley, 'Software Patents and the Return of Functional Claiming' 2013 *Wisconsin Law Review* 905–64, available at <http://ssrn.com/abstract=2117302 or <http://dx.doi.org/10.2139/ssrn.2117302> accessed 21 July 2022.

[17] EPO Board of Appeal Decisions: Computer program I/IBM (T1173/97) and Computer program II/IBM (T 0935/97). See EPO, Guidelines for Examination in the European Patent Office, G-II 3.6 (2019), available at <https://www.epo.org/law-practice/legal-texts/html/guidelines/e/g_ii_3_6.htm> accessed 24 August 2020.

[18] For example, *Aerotel Ltd v Telco Holdings Ltd* [2007] RPC 7; *Macrossan's Application* 2006 [EWCA], followed by *Symbian Ltd v Comptroller General of Patents* [2008] EWCA Civ 1066; *Halliburton Energy Inc's Patent* [2011] EWHC 2508 (Pat).

[19] Enlarged Board of Appeal Opinion G3/08. For commentary, see Freeman, 'Patentable Subject Matter: The View From Europe', note 16.

> taking that effect into account in the assessment of whether there is an inventive step has been developed, as laid out in T 154/04, *Duns*. While it is not the task of the Enlarged Board in this Opinion to judge whether this system is correct, since none of the questions put relate directly to its use, it is evident from its frequent use in decisions of the Boards of Appeal that the list of 'non-inventions' in Article 52(2) EPC can play a very important role in determining whether claimed subject-matter is inventive ... It would appear that the case law, as summarised in T 154/04, has created a practicable system for delimiting the innovations for which a patent may be granted.

In practice, as stated on various occasions by examiners of the EPO,[20] while they consider software-based inventions with technical effect as patentable subject matter, many software patent applications are being rejected on the basis of lack of novelty (the second hurdle, considering 'patentable subject matter' as the first) or lack of inventive step (the third hurdle).[21] In particular, mere computer- or software-based automation of constraints imposed by non-technical aspects—specifically those that are excluded by the EPC—notably mental acts, games, business methods, or methods for presenting information, are allegedly not being granted patent protection.[22]

10.2.2 In the US

In the US, for many years the leading decisions in the debate on software patentability were the US Supreme Court's decision in *Diamond v Diehr*[23] and subsequently *State Street Bank & Trust v Signature Financial Services*[24] where the Court of Appeals for the Federal Circuit held that a computerised algorithm for managing an investment fund structure constituted patentable subject matter which should be evaluated under the usual US tests of usefulness, novelty, and non-obviousness.[25] Subsequently, in *In re Bilski*, the Federal Circuit seemed to have

[20] See, e.g., EPO presentation by Eugenio Archontopoulos, 'Spot the Differences, A Computer-implemented Invention or a Software Patent?' (6th Annual Conference of the EPIP Association, Brussels, 2011) <https://www.researchgate.net/publication/230818897_Spot_the_difference_a_computer-implemented_invention_or_a_software_patent> accessed 16 June 2022.;

[21] In particular, features making no contribution to the technical character cannot support the presence of inventive step (*Comvik* (T0641/00) and *Duns Licensing* (T0154/04)). Also, Hanon 'What makes an Invention—How patent applications are examined at the European Patent Office', see note 20, and Archontopoulos, 'Spot the Differences, A Computer-implemented Invention or a Software Patent?' see note 20.

[22] *Ricoh Decision* T 03/0172; *Hitachi Decision* T 03/0258.

[23] 450 US 175 (1981).

[24] 149 F.3d 1368 (Fed Cir 1998) cert denied; 525 U.S. 1093 (1999).

[25] See Christopher Ogden, 'Patentability of Algorithms after State Street Bank: The Death of the Physicality Requirement' (2000) 10(82) *Journal of Patent and Trademark Office Society* 491–513.

begun to apply a more strict approach towards software patentability:[26] it found that a patent on a method of hedging financial risk in commodity trading claimed 'neither a new machine nor a transformation of matter', and thus was too abstract and non-patentable subject matter. However, the US Supreme Court then mitigated this analysis, to a certain extent, holding that the 'machine-or-transformation test' is not the only test for determining the patent eligibility of a process (but rather 'a useful and important clue ... an investigative tool, for determining whether some claimed inventions are processes under §101').[27] And in *Mayo Collaborative Services v Prometheus Laboratories, Inc*,[28] the US Supreme Court reaffirmed the judicially created exception that makes 'laws of nature, natural phenomena, and abstract ideas' ineligible for patenting, leading some to believe that there was an opening of the judicial 'door' to making the argument that software code is merely a series of mathematical algorithms and, as such, a description of abstract laws of nature.

The US Supreme Court's later decision in *CLS Bank v Alice Corp*[29] buttressed the importance of the non-software decision in *Mayo,* on software-related patentability determinations. Much like *Bilski*, *Alice* related to implementation of a business method: in *Alice*, a software-implemented system for managing escrow debt. In finding that particular invention patent-ineligible, the US Supreme Court stated that a two-step '*Mayo* framework' should be used in evaluating patent eligibility questions: the first step is to determine whether the challenged patent claim contains an 'abstract idea', such as an algorithm, method of computation, or other general principle; if it does, then the second step is to determine whether the challenged patent adds to the abstract idea an 'additional feature' that embodies an 'inventive concept'. [30] If so, the challenged claim is patent-eligible.[31]

[26] *In re Bilski* 545 F.3d 943 (Fed Cir 2008) (en banc). For comment, see, e.g., Dennis Crouch, 'In re Bilski: Patentable Process Must Either (1) Be Tied to a particular machine or (2) Transform a Particular Article' *PatentlyO* (30 October 2008) <http://www.patentlyo.com/patent/2008/10/in-re-bilski.html>> accessed 19 March 2021.

[27] *Bilski v Kappos*, No 08-964, 561 U.S. (2010). Comment by Crouch, 'In re Bilski', see note 26.

[28] *Mayo Collaborative Services v Prometheus Laboratories*, *Inc* 566 US (2012). Decision available at <http://www.supremecourt.gov/opinions/11pdf/10-1150.pdf> accessed 19 March 2021.

[29] 573 US 208 (2014).

[30] The addition of the 'inventive concept' test to patent eligibility determinations under *Alice* has been widely criticised as improperly conflating the non-obviousness requirement of 35 USC § 103 with the general patent eligibility requirements of 35 USC § 101. *See* Paxton Lewis, 'The Conflation of Patent Eligibility and Obviousness: Alice's Substitution of Section 103' (2017) 1 *Utah OnLaw: The Utah Law Review Supplement* Article 1, 13-32..

[31] The *Bilski-Mayo-Alice* triumvirate of US Supreme Court eligibility cases may not have entirely settled the question of how to evaluate whether a patent is directed to merely an 'abstract idea' and thus patent-ineligible. The Court of Appeals for the Federal Circuit's decision in *American Axle & Manufacturing, Inc. v Neapco Holdings LLC*, 939 F.3d 1355 (Fed. Cir. 2019) has been argued to import yet another statutory requirement—enablement under 35 USC § 112—into the 'abstract idea' analysis. David Taylor, 'Opinion Summary—American Axle & Manufacturing, Inc. v. Neapco Holdings LLC' *Federal Circuit Blog* (31 July 2020) <https://fedcircuitblog.com/2020/07/31/opin

Despite continued questions about the manner in which to evaluate the eligibility for patenting of software in the US, the number of 'software patents' being granted does not appear to have slowed down. This has also led to questions not only about whether many of the 'software patents' granted in the US—particularly those in the period between the *State Street* and *Bilski* & *Alice* decisions—are weak, if not trivial, and might ultimately fail upon a challenge as to eligibility under the current, or to be outlined in the future, test. In the interim, commentators have remarked upon the creation of patent 'thickets' of overlapping and poor-quality patents, which close down innovation and may make it difficult to operate in the software sector.[32]

So, all in all, current industry practice, the pressure from large software industry companies and other non-industry players such as non-practising entities, combined with the lack of resources and time for reviewing patents at the patent offices and the lack of access to relevant prior art in the field,[33] together mean that software patents have been and are still being granted over software implemented processes and methods on both sides of the Atlantic as well as in Japan, another key jurisdiction. Specific examples include security algorithms for encryption, audiovisual data codification and decodification ('codecs'), online data back-up, graphical user interface features, 'one-click' online shopping systems, frames for displaying information on computer interfaces, and the list goes on.[34]

ion-summary-american-axle-manufacturing-inc-v-neapco-holdings-llc/> accessed 28 August 2020. There seems to be some likelihood that the contours of the test for determining patent eligibility for claims argued to be directed to 'abstract ideas' have yet to be fully defined in the US, and there was thought to be a reasonably likelihood that the US Supreme Court might take up the *American Axle* case to further clarify patent eligibility—which might include clarifying patent eligibility for software in the US. Eileen McDermott, 'Solicitor General Tells SCOTUS CAFC Got it Wrong in American Axle, Recommends Granting' *IP Watchdog* (24 May 2022) https://www.ipwatchdog.com/2022/05/24/solicitor-general-tells-scotus-cafc-got-wrong-american-axle-recommends-granting/id=149248/> accessed 14 June 2022 (noting that the Solicitor General of the US—the office which offers the US Government's position on cases before the Supreme Court of the US—had requested that that court reexamine patent eligibility through that case). Much to the surprise of many who felt that the *American Axle* case was an ideal vehicle for further clarifying (or possibly changing) the patent-eligibility standards in the US, the US Supreme Court ultimately declined to review that decision. See Blake Brittain, 'U.S. Supreme Court rejects American Axle case on patent eligibility', *Yahoo! News* (30 June 2022) <https://news.yahoo.com/u-supreme-court-rejects-american-171958332.html> accessed 30 June 2022.

[32] Rosa Ballardini, 'The Software Patent Thicket: A Matter of Disclosure' (2009) 6(2) *SCRIPTed* <https://script-ed.org/wp-content/uploads/2016/07/6-2-Ballardini.pdf> accessed 19 March 2021, DOI: 10.2966/scrip.060209.207.

[33] Software patenting has a long history, dating back to at least the late 1960s. Gene Quinn, 'The History of Software Patents in the US' *IP Watchdog* (30 November 2014) <https://www.ipwatchdog.com/2014/11/30/the-history-of-software-patents-in-the-united-states/> accessed 19 March 2021. Nevertheless, for quite some period, there was little 'prior art' previously published in a meaningful manner —particularly in patent office databases—for disclosure against subsequent patenting.

[34] An interesting series of software patents can be found at the End Soft Patents wiki, 'Example software patents' <http://en.swpat.org/wiki/Example_software_patents> accessed 19 March 2021.

10.2.3 Differences with copyright

When a patent is granted on a software-based invention or CII, it doesn't just grant exclusionary rights over the exploitation of a specific implementation of that invention, but *any* implementation of the invention that meets all the elements of any claim in that patent—it protects the functional features of the 'invention', the underlying methodologies, in any manner or form of expression. This is in contrast with copyright protection, which only protects the expression embodied in either the source or binary code of the software.

This means that while copyright protection is generally weaker than patent protection, it is more specific, referring only to the concrete expression of the code developed by the programer. This has the advantages of providing legal certainty with regard to what exactly is prohibited or restricted by copyright, particularly verbatim copying,[35] and what is permitted—alternative or clean room development of similar functions, incremental development of additional functionalities, or complementary development of other programs using software interfaces and interoperability characteristics. Being more specific and restricted to expression, copyright enables a much broader range of alternative implementations and improvements of a same idea or function, through different algorithms, coding languages, or architectures.

There is a crucial distinction between the way patent and copyright concepts respond to the challenge free software poses. Copyright law is primarily intended to cover expressions, not ideas. So, if in a particular instance software copyright inhibits progress in making better, more reliable, or more effective software, the inhibition can be overcome: it is always possible for programers, with sufficient guidance and appropriate measures to prevent copying, to sit down and rewrite from scratch whatever program needs to be available in a freely modifiable version. This may be time-consuming, but it cannot be forbidden. Patent law, in contrast, prohibits anyone from practising the claimed subject matter of the patent without licence. It does not matter how you came by the idea the patent discloses, even if you invented it for yourself in complete ignorance of the patent and any prior art it references: without a licence you cannot implement, in *any* way, the claimed subject matter of what may be quite general claims.[36] This enables patent holders potentially to restrict competition by other developers wishing to implement similar functionalities in their own programs using completely different code expressions. Patents can also seem vaguer or less definite, particularly in the way software patents have been drafted in the time before the *Bilski*, *Mayo*, and *Alice* decisions in

[35] To a major extent, although there are always questions about non-verbatim copying and derivative works which the courts deal with on a fairly regular basis.

[36] Eben Moglen, 'Free Software Matters: The Patent Problem' *Moglen Law* (9 October 2000) <http://moglen.law.columbia.edu/publications/lu-05.html> accessed 19 March 2021.

the US. It is often quite difficult to determine exactly if the implementation of a software process may infringe an existing patent, as there is no way to 'clean room' develop code to avoid a patent. This creates significant legal uncertainty.

The law on software patents, unlike software copyright in jurisdictions like the European Union (EU), provides no exemption for interfaces. As an interface is a set of definitions or specification of a method or process (for using the program or data), it is particularly prone to being 'patentable'. So not only is there the potential for patents foreclosing specific computer-based processes but also there may be patents over software interfaces that may be required to connect with and use software processes.

Another significant difference between copyright and patents (relevant for Open Source) is the characteristics and structures of creation and ownership of rights: copyright in a software program belongs originally to its creator (or the company where the creator works), who has invested time and effort in developing the code, and the rights may be licensed or assigned, usually to someone who wishes to use or further develop the program. Thus, copyright rights are generally held by parties interested in exploiting the software. A patent is first owned by its inventor, who may or may not be a software developer. As there is not necessarily any 'software development' involved in inventing a process that may be embodied by software, the patent rights may be held by any party, who may or may not be interested in implementing the patented process or method, and in some cases may be held by a party interested in controlling or precluding the use by others of the patented process or method.

This situation is illustrated by what have been now called 'non-practising entities' (NPEs) (often pejoratively described as 'patent trolls').[37] These are persons or companies that do not have any particular interest themselves in exploiting the software that implements the patented processes, but only in asserting the patent rights against participants in the software industry interested in the invention, as a mechanism to extract royalty or other payments. NPEs also are less susceptible to external pressures that would otherwise forestall their use of patents to inhibit software use and deployment—because they have no business other than to assert patents, counter-assertions or business pressures are generally ineffective. While assertions of this sort are a legitimate function of patent rights, this creates a significant imbalance in the software sector and can constitute a major block on innovation.[38] This is not to say that there are not 'copyright trolls', monetising copyrights

[37] Wikipedia, 'Patent troll' <http://en.wikipedia.org/wiki/Patent_troll> accessed 19 March 2021.

[38] For commentary, see James Bessen, Michael Meurer, and Jennifer Ford, 'The Private and Social Costs of Patent Trolls' (19 September 2011) Boston University School of Law, Law and Economics Research Paper No. 11-45 <http://ssrn.com/abstract=1930272> or <http://dx.doi.org/10.2139/ssrn.1930272>.

through litigation.[39] We will comment further on this later, when looking at the interactions and frictions between Open Source and patents.

10.2.4 Patent remedies

The remedies available to patent holders in the case of infringement are important to understand the potential effect of patents against Open Source. National courts in Europe are competent to hear infringement cases and determine remedies of both the national equivalents of European patents and patents issued directly by their national offices. However, except for very limited circumstances, the national court's decision will only apply in its territory, and if the infringement occurs in several states, then proceedings would have to be brought independently in each country.[40] This is likely to change when the Unified Patent Court (UPC)[41] comes to fruition. The UPC will be a specialised patent court with exclusive jurisdiction for litigation relating to European patents and European patents with unitary effect (unitary patents). In practice, absent a UPC, Germany seems to be one of the favourite states to start infringement proceedings, as those proceedings are relatively cheaper and faster there (many decisions are made under the fast injunction

[39] In the early 2000s, SCO was accused of being a 'copyright troll' against UNIX and Linux. David Kravets, 'Copyright troll loses high-stakes Unix battle' *Wired* (31 March 2010) <https://www.wired.com/2010/03/unix-copyrights/#ixzz0yUsnFxzG> accessed 28 August 2020. More recently, an individual named Patrick McHardy has been accused of being a 'copyright troll' as the result of GPL violation lawsuits filed in Germany. Ieva Giedrimaite, 'Copyright trolling: Abusive litigation based on a GPL compliance' *The IP Kitten* (24 February 2019) <https://ipkitten.blogspot.com/2019/02/copyright-trolling-abusive-litigation.html> accessed 28 August 2020.

[40] It is possible to bring action against the defendant in its jurisdiction of residence and the local courts may in this case handle infringements across the relevant EU territories based on the origin of infringement with the defendant in its residential jurisdiction. There is also a practice in Dutch courts of granting cross-border injunctions in patent cases, although the circumstances under which can be done are likely limited to summary proceedings. Renaud Dupont, 'Cross-border injunctions are back in the Netherlands' *Lexology* (27 September 2011) <https://www.lexology.com/library/detail.aspx?g=2b5e8ef1-bf5a-46fd-8499-61f766c83424> accessed 29 August 2020. See also *Solvay SA v Honeywell Fluorine Products Europe BV*, Case C-616/10 (ECJ 12 July 2012).

[41] The Agreement on the UPC was endorsed by EU ministers in the Competitiveness Council on 10 December 2012 and by the European Parliament on 11 December 2012; because of Brexit and an adverse ruling from the German Federal Constitutional Court, the Unified Patent Court was for some time believed not to have achieve sufficient ratification to commence, and many predicted that it would not be instituted. James Nurton, 'German decision puts Unified Patent Court agreement in jeopardy' *IP Watchdog* (20 March 2020) <https://www.ipwatchdog.com/2020/03/20/german-decision-puts-unified-patent-court-agreement-jeopardy/id=120013/> accessed 29 August 2020. However, Germany eventually ratified the UPC, setting the UPC up to commence operation in 2022 or 2023—although there still remain questions as to whether the UK is required to ratify and participate in the UPC. Christoph Crützen, Benjamin Beck, and Maximilian Kücking, 'Germany Ratifies EU Unified Patent Court (UPC) Agreement, but Prospects for the UPC Remain Uncertain', *Mayer Brown blog* (18 August 2021) <https://www.mayerbrown.com/en/perspectives-events/publications/2021/08/ger-germany-ratifies-eu-unified-patent-court-agreement> accessed 14 June 2022.

procedure), something that has been seen in the case of the *Apple v Samsung* proceedings relating to Samsung's 'Galaxy' tablet.[42]

Remedies have been broadly harmonised across the EU through Directive 2004/48/EC of the European Parliament and of the Council of 29 April 2004 on the enforcement of intellectual property (IP) rights.[43] Remedies include both precautionary measures, such as preliminary injunctions and seizure, as well as permanent orders and monetary damages.

As the patent holder's main goal is to stop the infringing party's actions, it will mainly aim for preliminary and then permanent injunctions to cease the manufacture, distribution, commercialisation, and use of the infringing product. In addition, at the preliminary stage the patentee may request an order to seize or produce for audit products, tools (including computer equipment), production plants, books of account, invoices, and advertising materials, the latter in order to collect documentary evidence of the infringement and its extent; and a blocking order to stop imports at the national borders. In the extreme, a patentee may also request freezing the allegedly infringing party's bank accounts. Thereafter, when infringement is finally determined, the rights holder can request a declaration of the validity of the patent and the destruction of the infringing items.

If infringement is found, damages may be applied for to compensate for the infringing activities, either as accounts for profits made, monetary compensation for lost profit of the patent holder, or the fees the patentee would have charged for granting a licence (probably the preferred method, as proving lost profits or trying to work out the infringer's illegitimate profit made on the basis of the patented item, can be difficult).

We will see in the following section how difficult it is to apply these concepts in the Open Source software context. Not only is identifying infringers of a CII implemented in Open Source potentially unknown or difficult to identify or locate (assuming that the Open Source project is the 'person' infringing a third party's patent), but also it can be extraordinarily difficult to prevent distribution of intangible goods (that may infringe on patent rights) on the Internet.[44]

This is not the case when the software is embedded in hardware devices, such as smart phones, set-top boxes, or routers, where the patent holder may pursue any member of the supply chain (in particular the retailer and the importer) to obtain the injunctive relief and subsequent claim for damages. This is probably one of the

[42] See Chris Foresman, 'Apple stops Samsung, wins EU-wide injunction against Galaxy' *ArsTecnica* (9 August 2011) <http://arstechnica.com/apple/2011/08/samsung-facing-eu-wide-injunction-against-galaxy-tab-101> accessed 14 June 2022. Germany is a preferred venue, see comment by Kevin O'Brien, 'German Courts at Epicenter of Global Patent Battles Among Tech Rivals' *New York Times* (8 April 2012) available at <http://www.nytimes.com/2012/04/09/technology/09iht-patent09.html> accessed 19 March 2021.

[43] *Official Journal of the European Union* L157 of 30 April 2004.

[44] See, e.g., how OpenSuSE community deals with audiovisual codecs encumbered by patents: OpenSuSE wiki, 'Restricted formats' <https://en.opensuse.org/Restricted_formats> accessed 19 March 2021.

reasons patent litigation has been so popular in the mobile device industry, as there are specific goods or devices to identify for remedial action.

Thus, there are a series of reasons why patents are relevant to software, in particular their very existence with respect to software implemented inventions, their nature and scope, and their differences with copyright, many of which, as we will see next, enter into conflict with the principles and reality of Open Source.

10.3 Patents and Open Source Interactions

To understand the interactions between Open Source software and patents, we must briefly review the nature and characteristics of Open Source and its development process. As we will then see in this section, these are not particularly well-suited to the patent system (as legislated and practised), leading to a variety of areas and types of friction. In the next section of this chapter, we will look at how the Open Source community tries to deal with these frictions, both in the licensing regimes and in practice.

10.3.1 Development and Innovation in Open Source

Open Source is software that is distributed under an Open Source licence. These licences are broad, RF licences that allow all persons to use, copy, modify, and distribute the original code and its derivative works.[45] Thus Open Source is characterised by the granting to others of the ability to exploit the software, with access to its source code as a requirement to be able to enjoy those rights.

Any Open Source licence is in fact a practical expression of the ideals and objectives of the software creators, using copyright rights (and in some cases, patent rights) to allow and enforce openness and freedom with respect to the software code and the knowledge contained therein. Open Source licensing increases public accessibility to this knowledge. Under copyleft licences,[46] a sub-group of Open Source licences, this knowledge and these freedoms to exploit and innovate are guaranteed for all third parties through obligations to maintain the free software licensing terms in downstream distributions of the product and its derivative works.

In practice, this usually leads to a decentralised software development model, the ‘bazaar’, as Eric Raymond has called it,[47] whereby developers from all parts of the world may participate in and contribute to an Open Source project. These

[45] Stallman, ‘The Free Software Definition’, note 10. See also Open Source Initiative, ‘Open Source Definition’ available at <https://opensource.org/osd> accessed 19 March 2021.

[46] Stallman, ‘The Free Software Definition’, note 10.

[47] Eric Raymond, ‘The Cathedral and the Bazaar’ (2000) <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar> accessed 19 March 2021.

participants form what has generically been called the 'community' of the project, and these communities together form the 'Open Source community' or movement as a whole. These communities are extremely heterogeneous, including individual programers and users, institutions, companies, and public bodies, and can be formed by one or two persons, or a significant number of participants such as the Open Document or GNU/Linux communities.[48] The community participants, acting usually remotely over the web, maintain, develop, and correct the project software according to a roadmap that may or may not be an agreed 'master' document. In some communities, such as the Mozilla, Ubuntu, or Alfresco projects, the project may be led or structured by a foundation or corporate entity, which guides development and may exploit the software (or services based on the software) commercially.

Innovation in these communities is varied, either incremental—developers building on previous contributions made by themselves or other participants, or complementary—developing new functionalities and modules through standard and open interfaces. However, in all circumstances, innovation is based on the principles of freedom and openness: taking advantage of broad rights to copy, share, and improve the code, along with open access to the source and interoperability information of the project code.[49]

The certainty provided by the standardised copyright licensing terms established by the project Open Source licence provides reliability and trust among the participants, increasing network effects and providing a strong basis for further innovation.[50] In transaction cost analysis terms, this 'lowers the informational and transactional cost of licensing, as the terms are standard and transparent to all parties, so there is no information asymmetry and no need to negotiate terms'. [51]

10.3.2 Frictions with the patent regime: differences in concept

This form of innovation through sharing, however, runs counter to the justification for patent protection, which is based on the historical and theoretical foundation

[48] See, e.g., Linux Foundation Annual Report 2020, estimating 890,000 contributors, including 44,000 'core developers'. Linux Foundation, 'Annual Report 2020' (2020) <https://www.linuxfoundation.org/wp-content/uploads/2020-Linux-Foundation-Annual-Report_120520.pdf> accessed 19 March 2021.

[49] Chris diBona, 'Introduction' in Chris DiBona, Sam Ockman, and Mark Stone (eds), *Open Sources: Voices from the Open Source Revolution* (Sebastopol, CA: O'Reilly Media, 1999) 1–18.

[50] Notwithstanding the difficulties of interpreting certain licences in certain conditions, for example, the copyleft scope of the GPL. However, the most vibrant Open Source community, the Linux Community, uses the GPLv2 as its legal foundation, showing that this is not an impediment to innovation and sharing.

[51] Jason Schultz and Jennifer Urban, 'Protecting Open Innovation: The Defensive Patent License as a New Approach to Patent Threats, Transaction Costs, and Tactical Disarmament' (2012) 26 *Harvard Journal of Law and Technology* 1, 15.

of IP rights regimes, that of providing economic incentives to creativity and innovation through the artificial creation of exclusivity,[52] although this exclusivity does eventually end and the patent subject matter enters the public domain, upon expiration of a patent's term. Yochai Benkler, among others, has clearly argued that in the information society, as exemplified by free software production models, this justification is not necessarily correct, as there are (many) other incentives to innovation, including curiosity, need, benefits to reputation, the simple desire to share knowledge, or stimulating demand for a related product or service.[53]

Patents also offer the risk of over protection: going back to the historical debate of how to protect and incentivise the creation of software, there were arguments against the broad protection granted by patent rights over 'any' implementation of a particular process, its functionalities, its interoperability, and the impossibility of carrying out reverse engineering, as being too wide and hindering competition and innovation in this sector.[54] Recognising this, the copyright legal regime for software—at least in the EU—provides express exclusions for interoperability and reverse engineering to study the principles and ideas behind a software program, for example to be able to reproduce in a new manner its functionalities.[55]

This is particularly important for Open Source, one of whose main areas of development is the reverse engineering of proprietary software formats and functionalities, to create and distribute under Open Source licence terms both programs with similar features and software that is interoperable with proprietary formats (e.g., OpenOffice.org/LibreOffice or SAMBA).[56]

[52] See, e.g., Paul David 'Intellectual Property Institutions and the Panda's Thumb: Patents, Copyright, and Trade Secrets in Economic Theory and History' in Mitchel Wallerstein, Mary Mogee, and Robin Schoen (eds), *Global Dimensions of Intellectual Property Rights in Science and Technology* (National Academy Press: Washington, DC, 1993) 19–62; or Gillian Hadfield, 'The Economics of Copyright' (Columbia University Press: New York, 1992) 38 *Copyright Law Symposium* 1-46; reviewed in Christian Handke, 'The Economics of Copyright and Digitisation: A Report on the Literature and the Need for Further Research' (London: World Economic Press, 2010). For counter arguments, see Michele Boldrin and David Levine, *Against Intellectual Monopoly* (Cambridge: Cambridge University Press, 2008) esp. ch 7, 'Defenses of Intellectual Monopoly'.

[53] Yochai Benkler, *The Wealth of Networks: How Social Production Transforms Markets and Freedom* (New Haven, CT: Yale Press, 2006) at 63. Collaborative development models are also described in various articles in DiBona et al. (eds)*, Open Sources: Voices from the Open Source Revolution,* see note 49; and, e.g., Chris DiBona, 'Open Source and Proprietary Software Development' in Chris DiBona, Danese Cooper, and Mark Stone (eds), *Open Sources 2.0: The Continuing Evolution* (Sebastopol, CA: O'Reilly Media, 2006) 21–36.

[54] See debates of WIPO, Advisory Group of Governmental Experts on the Protection of Computer Programs, *Copyright* (WIPO's monthly bulletin) March 1971, 5–40; and WIPO Group of Experts on the Legal Protection of Computer Software, *Draft Treaty for the Protection of Computer Software* (Geneva, 13–17 June 1983).

[55] See WIPO Model Provisions for the Protection of Software 1983 and, e.g., EC Software Directive, arts. 5 and 6.

[56] Libre Office: <http://www.libreoffice.org> and Samba: <http://www.samba.org> accessed 19 March 2021.

In *SAS Institute v Worldwide Programming*,[57] the European Court of Justice (ECJ) reviewed the question of the protection by copyright of software functionalities, in the context of innovation and technical progress, concluding that:

> [o]n the basis of those considerations, it must be stated that, with regard to the elements of a computer program which are the subject of Questions 1 to 5, neither the functionality of a computer program nor the programming language and the format of data files used in a computer program in order to exploit certain of its functions constitute a form of expression of that program for the purposes of Article 1(2) of Directive 91/250.
>
> As the Advocate General states in point 57 of his Opinion, to accept that the functionality of a computer program can be protected by copyright would amount to making it possible to monopolise ideas, to the detriment of technological progress and industrial development.[58]

However, what is granted by the copyright regime (reverse engineering and interoperability), can be taken away by the patent regime. And although the copyright and patent regimes should ideally be complementary and non-exclusionary, an outcome in which one regime gives a right that the other regime takes away seems illogical taking into account that the objectives of the two regimes, to incentivise and reward creativity and innovation, are basically the same.

10.3.3 Patent frictions in practice

Not just on a theoretical basis but also in practice, there are a significant number of friction areas between the legal regime for patents, and Open Source and its production and distribution models.

First, as regards obtaining patents—if the Open Source community did ever want to patent inventive processes of a project—in environments where innovation is incremental and distributed throughout a community, it may be difficult if not impossible to determine who would qualify as an inventor. And who ultimately should be the beneficiary and rights holder of the patent rights resulting from community development? There is often no such figure or entity to hold them, other than all the individuals who contributed to the conception of the invention itself.[59]

[57] *SAS Institute Inc v World Programming Ltd*, C-406/10.

[58] ECJ decision C-406/10, paras. 39, 40.

[59] Joint ownership of a patent by a collection of developers can introduce complexities (or simplicities), depending on the jurisdiction in which the patent is granted. For example, in the US, all named inventors would have the right to exploit (use for their own purposes) the patent, including licensing it to others—including under an Open Source licence—without having to account (i.e. pay) any of the other inventors. This is not the case in other countries, including the UK (where consent is required from other inventors for an inventor to grant licences). See Raymond Millien, 'The Default Law of Joint

Second, from a risk analysis point of view, the risk of infringing copyright in software is far lower than the risk of infringing a patent. Copyright infringement can be avoided by implementing good development practices and (if need be) creating new and independent versions of copyrighted software. With regard to Open Source licensed code, it is in fact quite difficult to infringe copyright, as most exclusive copyright rights in the original code that you may be working on or with, are granted. Conversely, a patent over a software process can stop anyone from making, using, or selling the patented invention, even if there is no copying of the inventor's original software (if any). This means that it may be impossible to avoid infringing a patent regardless of how much care is taken, particularly essential patents on standards. In the early 2000s, there was at least one published assertion that the GNU/Linux operating system might infringe some 280 software patents,[60] although there was substantial debate about the meaning of that assertion.[61] What's more, the source code availability of Open Source allows a patent-based plaintiff to evaluate infringement easily, while a reverse-engineered patent infringement evaluation of binary code would be more difficult. 'Software patents are dangerous to software developers because they impose monopolies on software ideas. It is not feasible or safe to develop nontrivial software if you must thread a maze of patents.'[62] Moreover, it is argued that this situation is worse for Open Source than for proprietary projects.[63] As we have commented, Open Source is often developed by many people—volunteers—in 'open' communities. These communities rarely have any company or institution providing (legal or financial) support, and thus the individual developers might be more vulnerable to litigation. They certainly don't have the financial resources to cover the cost of dealing with patent issues, which can cost thousands if not millions of Euros. However, a counter-argument is that these individuals are not worth pursuing by patent holders, which may be one of the reasons that to date there are few if any patent-based cases against non-commercial Open Source projects.[64]

However, the counter to this is that any corporate end-users could be viewed as vulnerable to attack. While copyright focuses on the potentially infringing

IP Ownership' *IP Watchdog* (18 February 2016) <https://www.ipwatchdog.com/2016/02/18/the-default-law-of-joint-ip-ownership/id=66154/> accessed 19 March 2021; UK Patents Act 1977 (as amended) § 36-2(a) (1 October 2014).

[60] See Daniel Lyons, 'Linux Scare Tactics' *Forbes Magazine* (8 February 2004) <http://www.forbes.com/2004/08/02/cz_dl_0802linux.html> accessed 19 March 2021; and *Open Source Risk Management Position Paper—Mitigating Patent Risks* (2 August 2004).

[61] Steven Vaughn-Nichols, 'Author of Linux Patent Study Says Ballmer Got It Wrong' *EWeek* (19 November 2004) <https://www.eweek.com/servers/author-of-linux-patent-study-says-ballmer-got-it-wrong> accessed 29 August 2020.

[62] Richard Stallman, 'Europe's 'Unitary Patent' Could Mean Unlimited Software Patents' <http://www.gnu.org/philosophy/europes-unitary-patent.html> accessed 19 March 2021.

[63] Jason Morgan, 'Chaining Open Source Software: The Case Against Software Patents' (1999) <https://groups.csail.mit.edu/mac/projects/lpf/Patents/chaining-oss.html> accessed 19 March 2021.

[64] For more detail about patent litigation against Open Source, see section 10.6 later in this chapter.

copying, transformation, and distribution of software (thus acts carried out by persons in the software industry), any person who also *uses* software that infringes a patent is liable and can have monetary damages and an injunction awarded against them, regardless of whether they were aware of the patent or had any intent to infringe it, and regardless of whether they have any technical or other expertise in dealing with patents. This has a significant impact across industry, raising development expenses, and increasing legal risks and insurance premiums. This also hinders the uptake of the Open Source projects' output through fear of litigation, or making it more expensive by encouraging participants to take a royalty-bearing patent licence.

For a non-commercial Open Source project (and most commercial ones too), taking a patent licence can introduce difficulties. Patent licences and associated royalties are usually based on usage, and an Open Source project rarely if ever knows how its software is used, improved, or redistributed. In addition, in the event of using any Open Source under copyleft licences, in particular GPLv2, the patent licence would have to contemplate redistribution of the code unencumbered by any downstream patent restrictions so to enable the code to remain free; the patent holder would have to be willing to grant wide downstream user rights, something they are unlikely to be willing to do, absent any numerical data on usage.[65]

> We cannot just buy a patent license, because though free software isn't always free like free beer, it cannot exist at all unless it is free like free speech: everyone has to be allowed to take free code from one place and use it in another, or build on it, so long as she is willing to share and share alike.[66]

For certain copyleft licences, it can be difficult to achieve compatibility with copyleft licensing and receive the benefit of a patent licence, even a patent licence granted on RAND (reasonable and non-discriminatory terms),[67] although Red Hat has achieved it through its widely publicised agreement with Firestar. But Red Hat is in the unique position of having both the financial means and legal resources to negotiate such a licence.[68]

[65] See Section 7 of the GPLv2 available at <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html> accessed 19 March 2021. GPLv3, in contrast, has more limited restrictions upon further distribution in cases where the distributor has a patent licence allowing such distribution. See Section 11 of the GPLv3 available at <https://www.gnu.org/licenses/gpl-3.0.en.html> accessed 19 March 2021.

[66] Moglen, 'Free software matters: Patently controversial', see note 8.

[67] Discussed at length in Iain Mitchell and Stephen Mason, 'Compatibility of The Licensing of Embedded Patents with Open Source Licensing Terms' (2010) 3(1) *Journal of Open Law, Technology & Society (JOLTS)* 25–58 < https://jolts.world/index.php/jolts/article/view/57/100 > accessed 15 June 2022.

[68] See Red Hat press release, Red Hat Legal Team, 'Red Hat Puts Patent Issues to Rest' *Red Hat Blog* (11 June 2008) <http://www.redhat.com/about/news/archive/2008/6/red-hat-puts-patent-issue-to-rest> accessed 19 March 2021.

Often in cases of (corporate) patent litigation, the parties involved can and often do come to settlement through cross-licensing and patent peace agreements. These agreements are non-aggression agreements providing each party royalty-free access to a determined part of the other party's patent portfolio and often to specified products. This is prevalent in areas such as hardware manufacturing or biotech, and RF cross-licences are quite common in the computer hardware and software industry among proprietary companies. However, the nature of Open Source makes cross-licensing potentially non-viable; first, very few (if any) Open Source projects have any patents with which to 'trade' with a potential patent asserter. Second, there may not be a particular institution or entity with which to negotiate such an agreement—with the exception of corporate sponsored developments, such as Red Hat, which as we have mentioned, can and have negotiated patent licences; in addition, the GNOME Foundation recently negotiated a settlement of a patent assertion made against some of its Open Source.[69] Third, any potential legal entanglement due to software patents creates uncertainty and significant fear within the project community. Few Open Source projects are going to go near any patented technology or process—if they ever get to know about it—merely due to the risk of patent litigation and the transaction costs for dealing with the patent situation.

It has been argued, in the context of patents over standards, that from an economic perspective patent licences and royalties may be compatible with Open Source development models (this is fully discussed in Chapter 12): it is just a question of implementing an appropriate technological or business process for licensing and collecting the dues.[70] Indeed, there are Open Source projects such as Fluendo[71] whose very existence and business model lies in dealing with patents rights over audiovisual codecs, and interested third parties can purchase licences to these patent rights so as to implement and distribute proprietary patented codecs in Open Source multimedia environments. However, above and beyond the legal incompatibility when using copyleft licences, most non-commercial (and many commercial) Open Source projects are particularly incompatible with royalty-bearing technologies, since an essential characteristic of the project is to share the code easily among community participants (including users), and they have no visibility or control of downstream users. Requiring even minimal royalties would greatly hinder the freedom of developers to share and distribute the code they write.

This is reinforced by the sheer number of software-related patents that are applied for and issued annually (particularly in the United States), as well as the legal

[69] See section 10.6 later in this chapter.

[70] Jay Kesan, 'The Fallacy of OSS Discrimination by FRAND Licensing: An Empirical Analysis' (22 February 2011) *Illinois Public Law Research Paper* No 10–14.

[71] Available at Fluendo <http://www.fluendo.com> accessed 19 March 2021.

uncertainty about many of those that are issued (for lack of novelty, inventiveness, or patentable subject matter, as discussed earlier).[72] It would be impossible—if not counterproductive, as they could then be claimed to be knowingly infringing a patent, if subsequently litigated—for software developers to read through all the software patents relevant in their area of expertise (let alone 'all' software patents generally), and subsequently take an informed view on the validity, or not, of those patents.

Another significant area of concern for the Open Source community is the accumulation of patents in proprietary software companies. Usually, large companies like IBM use patents defensively. As they know that other companies in the industry will apply for patents, and then may sue for patent infringement in order to gain a competitive advantage, a company that wants to defend itself files for its own patents to use against its competitors. This either creates a massive patent war, such as that that has occurred in the mobile device industry,[73] or creates a *détente* or hold-off between the company and its competitors where each could sue the other in a similar way, so neither one does (and eventually they enter into cross-licensing agreements such as those mentioned earlier). However, members of the Open Source community have historically shown concern with large proprietary corporations asserting patent claims, directly or through associated patent assertion and licensing entities such as Intellectual Ventures,[74] to acquire a range of software patents that they can potentially use in the future to attack and try to restrict the development and distribution of Open Source software.

Finally, and this is linked to the previous point, we must mention NPEs.[75] These entities accumulate patents solely for the purpose of demanding patent royalties from third parties, and do not themselves 'practise' or implement their patents or for that matter conduct any business other than licensing and asserting their patents. They do not make, use, import, sell, or offer for sale anything that could be infringing, inoculating them against countersuits. There are a significant number of these entities, such as Acacia Research Group, or Intellectual Ventures, holding large portfolios of patents (Intellectual Ventures is alleged to hold over 30,000 existing patents).[76] While NPEs typically target their activities against the

[72] Ballardini, 'The Software Patent Thicket', 207, see note 32.

[73] Involving Samsung, HTC, Motorola, and Apple, among others. See Don Reisinger, 'A look back at the great Apple-Samsung patent war' *EWeek* (8 August 2014) <https://www.eweek.com/mobile/a-look-back-at-the-great-apple-samsung-patent-war/> accessed 19 March 2021.

[74] Dennis Crouch, 'Intellectual Ventures: Revealing Investors' *PatentlyO* (18 May 20122) <http://www.patentlyo.com/patent/2011/05/intellectual-ventures-revealing-investors.html> accessed 19 March 2021.

[75] See Brian Yeh, 'An Overview of the "Patent Trolls" Debate' (2012) Congressional Research Service, <https://sgp.fas.org/crs/misc/R42668.pdf> accessed 15 June 2022, for a good overview of this problem.

[76] Todd Bishop, 'Intellectual Ventures sues HP, Dell and others over patents' *Geekwire* (12 July 2011) <http://www.geekwire.com/2011/intellectual-ventures-sues-hp-dell-patents> accessed 19 March 2021.

products and services of commercial entities, in particular proprietary software companies with funds to pay for royalties, they have also targeted Open Source, both Open Source-based commercial entities such as Red Hat, who had to deal, for example, with Firestar,[77] and non-commercial Open Source foundations such as the GNOME Foundation, who had to deal with Rothschild Patent Imaging (see section 10.6 later in this chapter). As opposed to litigation against industrial entities, where (negatively) the threat of patent retaliation or (positively) the offer of a cross-licence may be made, it is nearly impossible to use such a strategy against NPEs, leaving only the expensive (prohibitively so, for Open Source communities) options of paying a royalty or challenging the validity or infringement of the alleged patents, or abandoning the allegedly infringing software altogether.

Thus, in the end the patenting regime for software serves to benefit nearly exclusively large (proprietary) software companies with economic resources to apply for, defend, and litigate software patents, potentially to the detriment of the Open Source communities who are behind many of the current innovations in information and communications technology (ICT), unless efforts are made to assist these communities with patent threats.

In summary, software patents are expensive to acquire and enforce, and outside most Open Source projects' economic capabilities. They are also considered philosophically, culturally, and politically anathema to many Open Source communities and their members, as a restriction on their innovation. In addition, there is a perception that many of the patents that represent a potential threat against Open Source may be of dubious validity, due to lack of novelty or inventiveness—particularly given the continued development of tests for abstractness in the US. Even when they appear to be acquired for 'defensive' or other altruistic purposes, there has been no guarantee against someone later 'weaponising' them for use in an offensive attack.[78]

This has led the Open Source community in many cases to reject the current legal regime whose uncertainty enables obtaining patent protection (in any form, even the allegedly 'highly filtered' protection granted by the EPO) for software, arguing on the one hand that the whole system is too expensive for Open Source projects and small software publishers to benefit from (if they wanted to) and, on basis of their own experience and that of the software industry as a whole, that copyright provides sufficiently strong protection for software and incentive to innovate and create more.

In a now often quoted memo, Bill Gates said in 1991: 'If people had understood how patents would be granted when most of today's ideas were invented, and had taken out patents, the industry would be at a complete standstill today.'[79] On this

[77] Floyd Marinescu, 'Red Hat Sued Over Hibernate 3 ORM Patent Infringement Claim' *Infoq* (30 June 2006) <http://www.infoq.com/news/RedHat-Sued-Due-to-Hibernate-3-O> accessed 19 March 2021, settled in 2008.

[78] Schultz and Urban, 'Protecting Open Innovation', see note 51.

[79] Bill Gates, 'Challenges and Strategy Memo' (16 May 1991) <http://en.swpat.org/wiki/Bill_Gates_on_software_patents> accessed 19 March 2021.

issue Richard Stallman stated in 2004: 'Software patents are the software project equivalent of land mines: each design decision carries a risk of stepping on a patent, which can destroy your project. Because every such patent covers some idea and the use of that idea, which by giving monopoly on patents inhibits the development of software.'[80]

10.4 How Open Source Deals with Patents

We now turn to see how the community has reacted to and deals with the several interactions and friction areas between patents and Open Source, and the perceived patent threat.

The Open Source community's actions in this respect can be divided into two types of action: preventive measures, to minimise the impact of software patents on software freedoms, and reactive measures, taking action to neutralise current patent threats to free software development.

10.4.1 Patent clauses in Open Source licences

The first and most 'structural' preventive measure to deal with software patents is the incorporation of patent-related terms in Open Source licences. As we noted in the introduction, an Open Source project's community norms and guidelines are reflected in the chosen licence terms: they set out the rules for participation, in particular for contributing to and using the project software. The community has leveraged the licences to set out rules regarding patent grants and non-assertion among participants.

10.4.2 First-generation Open Source licences

The first generation of Open Source licences, particularly the permissive licences such as the BSD and X11/MIT licences, did not expressly mention patent rights, though based on the wording of the licences there are arguments that either an express or at least an implicit licence is granted.[81] Some legal writers believe that implicit patent licences are uncertain and not binding (in particular when there is

[80] Richard Stallman, 'Fighting Software Patents—Singly and Together' (2004) <http://www.gnu.org/philosophy/fighting-software-patents.html> accessed 19 March 2021.

[81] Van Lindberg, 'OSS and FRAND', see note 3; Peterson, 'Why so little love for the patent grant in the MIT License?', see note 5.

no consideration), giving rise to questions regarding their scope or duration, the impact of combing potentially patented software distributed under these licences with other programs or hardware, and the creation of derivative works, or that the licences licence copyright rights only and no patent rights are conveyed.[82] This is not a happy situation with regard to legal certainty for the Open Source community, and while these licences are still popular, contributors and users with concern about potential patent assertions, or who own significant patent portfolios and wish to have greater certainty about which part of their portfolio is being licensed, may eschew these licences in favour of more recent versions with explicit patent provisions.

Where a company did want to use one of these more permissive licences (Google Inc, in this instance, with regard to WebM VP8 video codec technologies), it added a patent licence grant and peace terms in an additional clause, tying the patent grant to its implementation of the patent claims.[83] The impact of this is twofold: the code that Google has distributed is effectively granted under the MIT licence, a recognised and standard Open Source licence permitting easy use and adoption, while users of Google's version of the code are given comfort and protection as regards claims with respect to patents that Google and other contributors may hold in the codec.[84] Enhanced versions of the MIT and BSD licences—the Universal Permissive Licence[85] and the 'BSD+Patent' licence,[86]were also created to take the basic framework of the MIT and BSD licences and add to it an explicit patent grant.

The GPLv2, first published in 1991, included wording directed to patents, with a stated aim of making GPL'd software redistribution incompatible with software patents rights assertion—either by contributors or licensees of contributors. GPLv2 does not have an express patent grant or non-assertion covenant. While a licence by the original creator cannot take away patent assertion rights of a third-party patent holder (rights to restrict distribution and use of a software that embodies the patent for example against payment of a royalty), what it can do is prevent the redistribution of the original software at all if such distribution under the terms of the GPL2 is prevented by patents encumbering the software; hence the name of Clause 7 of GPLv2, 'liberty or death':[87]

[82] Heather Meeker, *The Open Source Alternative* (Trenton, NJ: John Wiley and Sons, 2008); Kappos and Harrington, 'The Truth About OSS-FRAND', see note 3.

[83] Google's WebM, 'Additional IP Rights Grant (Patents)' <http://www.webmproject.org/license/additional> accessed 19 March 2019.

[84] This of course does not guarantee that 'all' potential patent rights in the codec are licensed, as Google may not hold all those rights.

[85] Open Source Initiative, 'Universal Permissive Licence' <https://opensource.org/licenses/UPL> accessed 30 August 2020.

[86] Open Source Initiative, 'BSD+Patent Licence' <https://opensource.org/licenses/BSDplusPatent> accessed 29 August 2020.

[87] This phraseology is based upon a famous speech in early US history. 'Give me liberty, or give me death!' Wikipedia <https://en.wikipedia.org/wiki/Give_me_liberty,_or_give_me_death!> accessed 29 August 2020.

> If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.[88]

The GPLv2 also forbids imposing any additional restrictions (such as patent encumbrances) on the rights granted by the licence to the recipients of the software. If a distributor does so, for example by asserting patent rights, their licence under the GPL is terminated. This effectively means that a patent holder who distributes a software program based on GPLv2 code, embodying one or more of its patents, may no longer assert those patent rights against downstream licensees who redistribute that program onwards,[89] or who incorporate the program in their own product. What is more, this has the effect that if a GPLv2 licensee does get a third-party patent licence to exploit the software, then to be able to redistribute it they must effectively ensure that all downstream licensees are covered. This was made explicit in GPLv3, published in 2007,[90] and Red Hat achieved this in its agreement with Firestar (with respect to one of its Open Source programs, called Hibernate).[91]

10.4.3 Second-generation Open Source licences

As Open Source software and the Open Source licensing model gained more popularity into the late 1990s, and as simultaneously it became clearer that software patents would be found in jurisdictions around the world to satisfy the requirements of national law, there developed a desire for Open Source licences with clear and

[88] GNU Operating System, 'GNU General Public License, version 2, (1991) <http://www.webmproject.org/license/additional> accessed 19 March 2021.

[89] The impediment on patent assertion is based upon the theory that the 'liberty or death' provision of GPLv2 includes an implied patent licence. Richard Stallman, 'Why Upgrade to GPL Version 3' Free Software Foundation (31 May 2007) <http://gplv3.fsf.org/rms-why.html> accessed 29 August 2020. The extent to which an implied patent licence would be found in GPLv2, and of what scope that licence would have, is an unresolved issue which led to a more detailed, express, patent licence being included in GPLv3.

[90] GNU, 'GNU General Public License' <https://www.gnu.org/licenses/gpl-3.0.html> accessed 19 March 2021.

[91] Maureen O'Gara, 'Red Hat Settles Patent Claims Against It' *DZone* (11 June 2008) <https://dzone.com/articles/red-hat-settles-patent-claims-> accessed 19 March 2021.

express terms around patent rights. One of the first organisations to take on this issue was Netscape, which was considering freeing its 'Navigator' web browser in 1998. That browser was released under the Netscape Public Licence[92] (later migrated into the Mozilla Public Licence 1.1), which included express patent provisions. Since that time, most newly created and OSI-approved Open Source licences also include an express patent licence grant of some scope.

The development of patent provisions in second-generation Open Source licences generally addresses two separate, but arguably related, issues. First, they grant an express patent licence to patent rights that the initial developer, or any contributor to the project, may have in their contribution. These express patent licences are in a variety of different forms, and each have differently expressed language, so determining exactly which patent rights are granted, and by whom, and for what, requires detailed analysis of the particular licence and the particular grant. Second, many—but not all—patent provisions in second-generation Open Source licences provide for defensive patent grant suspension (sometimes referred to as 'patent retaliation'), specifying conditions under which the express patent grant from authors or contributors is terminated or suspended in the event of a party that has received a licence initiating some form of patent litigation or other patent assertion with respect to the software.[93]

The ASF 2.0 License (2004)[94] provides a patent provision template that can serve as a model for an appropriate express patent licence grant, as well as a defensive patent grant suspension. The ASF 2.0 licence includes an express patent licence from each contributor to 'make, have made, use, offer to sell, sell, import, and otherwise transfer the Work'. This grant covers the contributor's contribution by itself, or when that contribution is combined with the software to which it is contributed. Similarly, the Mozilla Public License (MPL) 2.0 (2012)[95] contains an express patent grant covering the present and future patents rights of a contributor for the 'making, using, selling, offering for sale, having made, import, or transfer of either [the Contributor's] Contributions or its Contributor

[92] The Mozilla Foundation 'Netscape Public License 1.0' <https://website-archive.mozilla.org/www.mozilla.org/mpl/mpl/npl/1.0/> accessed 30 August 2020.

[93] The particular scope of the defensive patent grant suspension is important in evaluating whether a licence containing it may properly be considered an Open Source licence. Facebook, as one example, created a licence which included a defensive patent grant suspension provision that suspended the express patent grant in the event of any patent assertion against Facebook, whether or not that assertion related to the software licensed under that grant. This provision was roundly criticised as being non-reciprocal and was later withdrawn by Facebook. Sarah Gooding, 'Facebook to Re-license React after Backlash from Open Source Community' *WordPress Tavern* (25 September 2017) <https://wptavern.com/facebook-to-re-license-react-after-backlash-from-open-source-community> accessed 30 August 2020.

[94] The Apache Software Foundation, 'Apache License, Version 2.0' <http://www.apache.org/licenses/LICENSE-2.0.html> accessed 19 March 2021.

[95] Mozilla Foundation, 'Mozilla Public License Version 2.0' (MPLv2.0) <http://www.mozilla.org/MPL/2.0> accessed 19 March 2021.

Version',[96] and excludes deletions from, or modifications made to, the code, or combinations of the code with other software or devices, or the code in the absence of the contribution by that particular contributor.[97]

GPLv3 (2007) also has an express patent grant; Section 11 provides that '[e]ach contributor grants you [the user] a non-exclusive, worldwide, RF patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version'. The 'contributor version' is defined as '[any copyrightable work licensed] or a work on which [that copyrightable work] is based' which a copyright holder authorises use under the GPLv3 licence. 'Essential patent claims' in GPLv3 are defined as:

> all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, 'control' includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

The last point regarding 'control' is interesting, as in practice it permitted the flexibility for Red Hat to acquire downstream patent sublicensing rights from Firestar, so as to ensure valid onward GPL-based licensing of the Firestar patents to which Red Hat received a license. GPLv3 also allows alternative mechanism to allow a distributor of GPLv3 code to receive the benefit of a patent licence yet ensure that the source code remains available to the public.[98] GPLv3 has another patent-related requirement, drafted in response to a transaction between Microsoft and Novell,[99] which was designed to prevent unusually structured business deals believed to be a 'work around' to the concept of 'liberty or death'.[100]

[96] MPLv2.0, see note 95, Section 2.1(b). The 'Contributor Version' in this section is defined as 'the combination of the Contributions of others (if any) used by a Contributor and that particular Contributor's Contribution', similar to the way the Apache 2.0 licence covers combinations.

[97] MPLv2.0, see note 95, Section 2.3.

[98] 'If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients.' GPLv3, Section 11.

[99] Cath Everett, 'Inside the Microsoft-Novell deal' *ZDNet* (30 April 2007) <https://www.zdnet.com/article/inside-the-microsoft-novell-deal/> accessed 30 August 2020.

[100] This is reinforced by a paragraph in cl. 11 of the GPLv3 that provide for this very situation: If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorising them to use, propagate, modify or convey a specific copy of the covered

These clauses in each of the above-discussed licences, as well as many other licences approved by the OSI since 1998, ensure that users of software under these licences get the specified patent rights from the upstream contributors to the work. This does not mean that use of the software is free of patent risks, as third parties may have patent rights over the work and may not have granted the user any licence, and in many cases, subsequent changes made to the program after distribution by a patent holder may be unlicensed. Nevertheless, the user is protected from patent claims by the contributors, who—if the contribution is of original code—are usually the persons most likely to have any patent rights in that contribution.

10.4.4 'Patent defensive suspension' clauses

Patent defensive suspension clauses come in several different 'flavours', depending on the scope and conditions for triggering the clause. These provisions are often structured as a condition of the original licence grant—either just the patent grant, or all grants, including copyrights. Most are structured to protect the specific software to which a patent holder is licensed; a few against any suits based on patent rights over any software, not just the licensed software, although these broader provisions are now looked upon as non-reciprocally discriminatory and violative of Open Source Definition 5.[101] The provision may also revoke patent rights, or all rights granted under the Open Source licence. In Table 10.1 we will look at four licences, chronologically the MPLv2.0, Apache v2, EPLv2, and GPLv3.

What do these provisions achieve? On the one hand, as noted, Open Source participants using software under these licences have a certain degree of safety from

work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it'. The overall aim of this is to ensure a level playing field, and guarantee freedoms for the whole chain of licensees taking a copy of the code under the GPL. 'You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.' This provision was specifically aimed at the Microsoft/Novell transaction.

[101] OSD 5 states that an open source licence must have 'No Discrimination Against Persons or Groups'. Open Source Initiative, 'The Open Source Definition' <https://opensource.org/osd> accessed 30 August 2020. Any provision that takes licences away from entities asserting patents outside of the particular project to which a licence is granted is believed, by many, to violate this non-discrimination provision.

Table 10.1 Comparison of Defensive Suspension Clauses

Mozilla 2.0: Section 5.2	
If You initiate litigation against any entity by asserting a patent infringement claim (excluding declaratory judgment actions, counter-claims, and cross-claims) alleging that a Contributor Version directly or indirectly infringes any patent, then the rights granted to You by any and all Contributors for the Covered Software under [the copyright and patent licence] Section . . . of this License shall terminate.	This clause is of a scope relatively common for defensive suspension clauses, although it does include a suspension not only of patent licences, but other licences as well in the event of a patent assertion against the software. Note that it does allow such an assertion in the form of a counter claim or cross-claim (a claim that is filed in response to an initial claim against the patent asserter), which may provide for some litigation strategy gaming tactics to retain the benefit of the licences but still assert patents against the software.
Apache 2.0: Section 3	
If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.	The defensive suspension clause in Apache 2.0 licence is similar to Mozilla 2.0, although it only revokes potential patent right grants; it does not purport to terminate any copyright licence. It also does not exclude cross-claims or counterclaims, like Mozilla 2.0.
Eclipse Public License 2.0: Section 7	
If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) [the patent licence Section] shall terminate as of the date such litigation is filed.	This section is similar to Apache 2.0, in that it only suspends patent licences, and, like Apache 2.0, does not exclude cross-claims or counterclaims, thus preventing potential litigation strategies to preserve the licence grant while still asserting patents. Unlike the Common Public Licence, a predecessor of Eclipse which has been deprecated, it does not attempt to suspend patent licences for assertions against other software.
GPLv3: Section 10	
You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.	GPLv3 maintains similar 'liberty or death' provisions as its version 2, commented on in the introduction (now called 'No Surrender of Others' Freedom' clause), and includes this patent peace clause. Breach of this undertaking (not to initiate patent-based litigation with respect to the software in question) would mean breach of the licence, and revocation of all licence rights (both copyright, and patent) subject to the reinstatement provisions (e.g. if the litigation is withdrawn). GPLv3 does not limit this litigation against 'developers' but would cover litigation against 'any entity', similar to the Apache and Eclipse licences commented on earlier.

patent-related threats from upstream contributors as a result of the express patent licence grants, and downstream licensees as a result of the patent defensive suspension provisions; this provides a degree of 'patent peace' among community participants. The more participants involved in the community, particularly large patent-holding entities, the greater the peace, and all the more so if the licence is copyleft, and thus maintains the same licensing terms downstream and throughout the community of users. This contributes to the ideals of the Open Source community, of providing safe access to knowledge of Open Source technologies, and derivative works thereof, and freedom to innovate. 'Licensees and their sublicensees should not be able to benefit from Open Source while at the same time forcing the licensor to pay royalties for patents embodied in that very software.'[102] However, the scope of this protection does vary, and it is important (especially for patent-holding users or contributors to Open Source projects) to understand the scope of the express or implied patent licence clauses as well as defensive suspension provisions, and how they interact with their patent portfolios. These clauses may discourage patent holders from participating in communities, either because the patent licence grants are too broad—or too indeterminate—or because the defensive suspension provision curtails their ability to assert their own patent portfolio against entities towards whom they wish to maintain a strong patent position. An example of this is where a company drafted a modification to the MPL1.1 in order to protect their portfolio, rather than use the standard version of the licence,[103] or in the case where additional patent provisions have been appended to existing Open Source licences.[104]

In addition, there may be questions of validity of parts of these clauses. First, with regard to 'future' acquired patents and patent rights, and second, with regard to extending the benefits of the clauses to non-licensees, or extending the obligations either to future users (holding other patents) of the Open Source technologies, or future holders of relevant patents (for example, through acquisition), who may be able to argue they are not party to the original bargain. This may be a question of privity of contract, if licences are deemed to be contracts in this respect; though for licences that are considered to be unilateral authorisations (and not contracts), the provisions would only be effective against licensees (i.e. users) of the code, as a condition of the licence grant.

[102] Lawrence Rosen, 'Dealing with Patents in Software Licences' *Linux Journal* (1 January 2002) <http://www.linuxjournal.com/article/5575> accessed 19 March 2021.

[103] MXM Public license submission, OSI Review, 'For approval: MXM Public license' (8 April 2009) <https://lists.opensource.org/pipermail/license-review_lists.opensource.org/2009-April/000722.html>, commented on by Glyn Moody 'Should an Open Source Licence Ever Be Patent-Agnostic?' *Linux Journal* (9 April 2009) <https://www.linuxjournal.com/content/should-open-source-licence-ever-be-patent-agnostic> both accessed 19 March 2021.

[104] David Thompson, 'Reading the Fine Print in Facebook React's Open Source License' *White Source* (17 May 2017) <https://resources.whitesourcesoftware.com/blog-whitesource/reading-the-fine-print-in-facebook-react-s-open-source-license> accessed 30 August 2020.

10.4.5 Open Source as prior art, peer to patent, and defensive publication

Another way of the dealing with the negative impacts of software patents in a preventive manner is to help avoid those patents being granted *ab initio*. Although efforts to get software processes—CIIs—totally excluded (in theory and in practice) from patentability have largely been unsuccessful in most of the jurisdictions of the world, there have been other projects that have claimed to try to reduce, during the patent examination process, poor-quality applications on the basis of lack of conformity with the main requirements for patentability: novelty and inventive step.

One criticism of the patent application examination process, in terms of quality, is that patent examiners rely heavily on databases of issued patents and published patent application, and occasionally scientific publications, to discover prior art. This means that a significant amount, if not all, of previously published software and software-related documentation—both proprietary and Open Source—may not be taken into account during the prior art search stage of the examination process.[105]

Open Source as Prior Art was a project launched in 2005 as an initiative to enable Open Source software repositories to be considered during this prior art search stage, 'improving accessibility by patent examiners and others to electronically published source code and its related documentation as a source of prior art'.[106] Unfortunately, software in online repositories is not published in a manner that can easily be mapped against the way patent applications describe the claimed methods or processes. To ensure such software is taken into consideration, it needs to be time stamped, documented, and ideally categorised or described in a manner that can be searched. While this aim was laudable, in practice it has been found to be particularly difficult and time-consuming, so it seems the project is currently inactive.

In another attempt at improving patent quality, 'Peer to Patent' was a project launched by the US Patent Office (USPTO) together with New York University Law School, aimed at taking advantage of the software community to supply the USPTO with information and discussion relevant to assessing the claims of patent applications during the examination process, opening this process to public participation and 'community reviewing'. The goal of this project was to help third parties identify, submit, and rank prior art that is relevant to a patent application. The results of the initial phases of this project resulted in several patent applications being rejected or narrowed as a consequence of peer reviewing.[107]

[105] 'Do USPTO examiners search open-source codebases?' *StackExchange* (22 September 2012) <https://patents.stackexchange.com/questions/401/do-uspto-examiners-search-open-source-codebases> accessed 31 August 2020.

[106] The Linux Foundation, 'Open Source as Prior Art (OSAPA)' <https://wiki.linuxfoundation.org/osapa/start> accessed 19 March 2021.

[107] See results commented on by Andrea Casillas, 'Peer to Patent Pilot 2 Results' <http://www.slideshare.net/acasillas11/peer-to-patent-pilot-2> accessed 30 August 2020.

While there have been several criticisms,[108] and although the project was eventually discontinued, Peer to Patent has been seen as one of the factors leading to the creation of certain new processes for improving the quality of patents under the US America Invents Act of September 2011,[109] notably the possibility for third parties to file pre-issuance submissions,[110] prior art something similar to the observations phase of European Patent applications.[111] In addition, the Peer to Patent project has highlighted the need to take into account all prior art, not just in theory but also in practice, that is relevant to the patent applications that a patent office is reviewing: websites, journals, textbooks, software development, user manuals, and other non-patent databases. Community involvement and online discussion also helps find this information. This has a positive economic effect, as avoiding *ab initio* the granting of poor-quality and/or invalid patents is significantly cheaper than a re-examination or post-grant review processes, or invalidity procedures before the courts. Ideally, prior art submitted in this way would gradually reduce the ability of non-practising entities holding poor-quality patents to threaten Open Source projects.

As a third leg in the strategy for avoiding 'bad patents', *defensive publication* is coming to be seen as one of the most efficient and effective measures. IBM, for forty years, produced a publication of inventions, which it developed but did not seek to patent, as a mechanism for establishing prior art that might prevent others from later attempting to patent the same or similar technology.[112] Linux Defenders,[113] a program for defending the Linux operating system and the Open Source community as a whole against patent concerns and threats, and which also supported the Peer to Patent project, was an initiative to support defensive publication, by directing to a website 'Technical Disclosure Commons',[114] designed as a repository for individuals to make dated publications of technology disclosures for prior art purposes.

The Technical Disclosure Commons site provides a mechanism for developers and creators to submit a publication that is date-stamped so as to establish its public disclosure date for prior art purposes, with the goal that patent examiners and patent challengers may review and use these disclosures as prior art. The publications are posted to the IP.com prior art database, which allows patent offices worldwide to include these publications in their patent searches.

[108] Summarised at Wikipedia, 'Peer-to-Patent Criticisms' <http://en.wikipedia.org/wiki/Peer-to-Patent#Criticisms> accessed 30 August 2020.

[109] HR 1246 (112th), now Public Law 112–29, *Statutes at Large*, 125 Stat. 284 through 125 Stat. 341 (2012).

[110] 35 USC 122(e).

[111] WIPO has also taken up this idea for PCT applications, WIPO, 'Patent Cooperation Treaty (PCT) Working Group' (14–18 June 2010) <http://193.5.93.80/edocs/mdocs/pct/en/pct_wg_3/pct_wg_3_6.pdf> accessed 30 August 2020.

[112] 'IBM Technical Disclosure Bulletin' Wikipedia <https://en.wikipedia.org/wiki/IBM_Technical_Disclosure_Bulletin> accessed 30 August 2020.

[113] Linux Defenders: <http://linuxdefenders.org> accessed 30 August 2020.

[114] Technical Disclosure Commons <https://www.tdcommons.org/> accessed 30 August 2020.

10.5 Patent Busting and Patent Pools

The measures described earlier to make available and accessible more prior art so that those patents that are granted truly meet the tests of novelty and non-obviousness/inventive step are aimed at preventing the granting of poor-quality patents, not only in the software sector but in all technology fields. Another question for the Open Source community has been: what can be done about existing poor-quality patents that can be used to threaten the Open Source—and indeed proprietary software—communities and result in claims for unreasonable patent royalties or potentially injunctive remedies to stop distribution? This is a question of 'problem containment' and the strongest proposals so far focus on post-grant patent review, and creating defensive patent pools to protect specific areas of technology. Notably, these proposals are centred in the US, where the software and business method patent problem is believed to be most acute.

As regards patent review, there have been several community initiatives: one was the Linux Defenders project called 'Post-Issue Peer to Patent' which was designed to solicit prior art contributions from Linux and the broader Open Source community to permit the invalidation of previously issued patents that were issued in error because of the patent office's lack of awareness of relevant prior art. Like many of the initiatives discussed earlier, this project is no longer operational. Another is the 'Patent Busters' project, launched in 2004 by Electronic Freedom Foundation (EFF),[115] which organised collaborative community efforts to challenge existing patents that it had pinpointed as being particularly harmful to innovation. It then filed challenges to those patents it determined were not properly granted, which it had done with a certain degree of success.[116] The Patent Busters project does not appear to have taken any steps to 'bust' a patent since approximately 2016. The Public Patent Foundation (PUBPAT)[117] ran a similar project, with the aim of challenging through post-grant challenges US patents believed to be invalid. This project worked in all areas of technology, not just software,[118] although its activities have not been updated since 2015.

These actions have been supported in the US by the introduction in 2011 of post-grant review processes under the America Invents Act (AIA). One part of this legislation allows third parties to submit 'post-grant review' invalidity challenges

[115] EFF, 'Patent Busting Project' <https://www.eff.org/issues/patent-busting-project> accessed 16 June 2022.

[116] EFF at one time listed ten patents that challenged or wished to challenge under this project, with various degrees of success including complete invalidation ('busted'), narrowing, or some form of post-issuance reevaluation being initiated by the USPTO. Wikipedia, 'Patent Busting Project', <https://en.wikipedia.org/wiki/Patent_Busting_Project> accessed 16 June 2022.

[117] Public Patent Foundation, 'Undeserved Patents and Unsound Patent Policy Harm the Public' <http://www.pubpat.org> accessed 31 August 2020.

[118] Successes are listed at Public Patent Foundation, 'Protecting the public domain' <http://www.pubpat.org/Protecting.htm> accessed 31 August 2020.

of a recently granted patent, within nine months of issuance.[119] Grounds for invalidity include lack of novelty, obviousness, as well as non-compliance with description, enablement, or patent eligibility rules. Another part of this legislation allows third parties to submit '*inter partes* review' invalidity challenges at any time after 9 months from issuance, but only challenges for lack of novelty or obviousness based on patents or printed publications.[120] In the past, prior to the AIA, the only mechanism for patent challenges—the filing of an *ex parte* or *inter partes* re-examination—had to be based upon prior art patents or printed publications. The challenges represent a potentially cheaper mechanism for contesting the validity of an issued US patent than litigation, although the filing fees alone—over US $40,000.00—and the length and complexity of the procedures mean that such challenges can often mean costs in the hundreds of thousands of US dollars.[121]

One other mechanism to address patent threats is the creation of a defensive patent pool. The Open Invention Network (OIN),[122] controls a patent pool and has the mandate to defend Open Source—as defined in a Linux System Definition which began with core Linux but which today includes over 2,000 other Open Source packages—from patent attacks. It was launched in 2005, by six founding companies[123] and has received investment from four additional large technology industry participants[124] as well as its founders. OIN is free to join and works, at its simplest, on the basis of a mutual hold harmless, or commitment not to sue, amongst its 3,500 licensees, each of whom, like its founders, sign up to the same non-negotiable licence terms.

OIN has so far acquired a large (1,500+) portfolio of patents purchased at a cost in excess of US $100 million, 'all available royalty-free to any company, institution or individual that agrees not to assert its patents against the Linux System'. OIN will therefore buy patents (i) to stop them falling into the hands of non-practising entities, who might otherwise assert them against Linux-based companies;[125] and (ii) to provide a portfolio of patents that can be asserted against companies that attack Linux.[126] In fact, OIN partnered with Allied Security Trust to intercept

[119] 35 USC 321.

[120] 35 USC 311.

[121] Challenges to patents in the US based on prior art patents or printed publications using *ex parte* re-examination continue to be available, 35 USC § 302, and are likely much cheaper, but these proceedings can often be one-sided in favour of the patent holder and therefore generally are only used when the prior art is particularly strong.

[122] OIN <http://www.openinventionnetwork.com> accessed 31 August 2020.

[123] IBM, Phillips, NEC, Sony, Novell, and Red Hat.

[124] Canonical, TomTom, Google, and Nissan.

[125] See, e.g., OIN's purchase of twenty-two Silicon Graphics patents that Microsoft placed with Allied Security Trust to sell: Paula Rooney, 'OIN Outmanuevers Microsoft, Buys Linux Patents' *ZDnet* (9 September 2009) <http://www.zdnet.com/blog/Open Source/oin-outmanuevers-microsoft-buys-linux-patents/4800> accessed 31 August 2020.

[126] See, e.g., OIN's transfer of four patents to Salesforce.com after Salesforce.com was sued for patent infringement by Microsoft: Florian Mueller, 'The OIN gave Salesforce.com four patents to assert against Microsoft' *Fosspatents* (31 May 2011) <http://www.fosspatents.com/2011/05/oin-gave-salesforcecom-four-patents-to.html> accessed 31 August 2020.

Microsoft patents that were alleged to read on Open Source functionality and avoid those patents and associated claim charts from being 'washed' through Allied Security Trust (AST)—where they could have been licensed to AST's members before being passed to an NPE to have the claim charts enforced through litigation.[127]

Because all of the patents of all of the members of OIN are in effect licensed RF to all the other members in relation to the Linux System, that equates to a collective patent portfolio of over an estimated 350,000 patents and applications pledged not to be asserted against the Linux System software.

OIN acted successfully to convey patents from its extensive portfolio to Salesforce.com when it was sued for patent infringement of FAT filesystem patents by Microsoft. Rather than expose itself to a potential injunction, the counterclaim by Salesforce of the patents received from OIN precipitated a rapid settlement by Microsoft.[128] In addition, in at least one other action that has been made public, when TomTom was also sued by Microsoft over exFAT filesystem patents the spectre of OIN's conveyance of patents to TomTom coupled with TomTom's own patents that were used in the actual counterclaim was sufficient to trigger a settlement for a fraction of the original damage claim.[129] While there is little statistical data available regarding patent threats and assertions, the fact that OIN has routinely provided prior art to companies in the Open Source community at risk of, or actively in, litigation indicates that OIN's involvement may serve a useful vehicle to reduce patent threats in core Linux and the adjacent Open Source software space.

The foregoing NPE interventions notwithstanding, OIN historically was designed to primarily work to mitigate practising entity patent risk but since Microsoft became a member of the OIN Community in late 2018,[130] OIN has pivoted to put increasing focus on mitigating NPE risk. In addition to working with the Open Source technical community to identify prior art to be shared with Community members who are at risk or in litigation, OIN has also joined with the Linux Foundation, IBM, and Microsoft to found and fund the Unified Patents' Open Source Zone to enable the mitigation of risk from NPE-owned patents that read on Open Source functionality.

OIN is a defensive entity and not an assertion entity; that is, it has not itself commenced litigation against companies attacking Open Source using its existing patent portfolio, although OIN has sold hundreds of patents to companies in

[127] Nick Wingfield, 'Group of Microsoft Rivals Nears Patent Deal in Bid to Protect Linux', *Wall Street Journal* (8 September 2009) <https://www.wsj.com/articles/SB125236988735891147> accessed 19 August 2022.

[128] Florian Mueller, 'The OIN Gave Salesforce.com Four Patents to Assert against Microsoft', *FOSS Patents* (31 May 2011) <http://www.fosspatents.com/2011/05/oin-gave-salesforcecom-four-patents-to.html> accessed 19 August 2022.

[129] See comment by Software Freedom Law Center, 'Settled, But Not Over Yet' (30 March 2009) <http://www.softwarefreedom.org/news/2009/mar/30/settled-not-over-yet> accessed 31 August 2020.

[130] Navneet Akash, 'Microsoft Joins OIN, Makes 60,000 Patents Open-Source', *C#Corner* (12 October 2018) <https://www.c-sharpcorner.com/news/microsoft-joins-oin-makes-60000-patents-opensource> accessed 15 June 2022.

litigation or at risk from operating companies poised to assert patents containing Open Source-related claims.

In addition to OIN, it has been suggested that an assertion entity (Fair Troll) acting on behalf of the Open Source community to recoup sums paid in patent licensing might have attractions.[131] Given that the Open Source community in general has been vocally anti-software patent, creation of such an entity with community support seems unlikely.

In March 2013, Google published a proposal to establish and standardise defensive patent pools, with the objective of reducing patent litigation concerns, particularly by NPEs.[132] One particular part of this proposal that eventually came into fruition was the proposal of a Licence on Transfer (LOT) regime whereby companies would band together and commit that they would grant one another licences to their patents, even if those companies did not have in place any existing patent licence arrangements between them, in the event that one of their patents was sold or otherwise transferred to an entity that might be non-practising. This resulted in the formation of the LOT Network, in 2014, to achieve exactly this result.[133] The network has grown to over 1,000 participants in a relatively short period and, though the benefits of this network extend only to members and it is not an Open Source-specific solution in the manner of OIN, companies active in Open Source can gain protection from NPE risk by joining LOT; large companies pay a modest annual fee while companies below a certain size receive complimentary membership.

Finally, during the 2000s, various companies made patent pledges in favour of individuals and groups working on Open Source—unilateral promises not to assert patents against developers, provided that certain conditions are met. These pledges are intended to operate as an enforceable covenant not to sue, and equitable estoppel should preclude the patent holder from bringing suit against those within the safe harbour defined by the pledge.

Notable patent pledges include Red Hat,[134] Nokia,[135] and IBM.[136] Note also that many of these pledges may have been expanded by the joining of some of the

[131] Florian Mueller, 'The DPL and the "Fair Troll" business model: make money fighting patents with patents' *FOSS Patents* (18 May 2010) <http://www.fosspatents.com/2010/05/dpl-and-fair-troll-business-model-make.html> accessed 31 August 2020.

[132] Eric Schulman, 'Working together to reduce patent litigation' *Google Public Policy Blog* (12 March 2013) <http://googlepublicpolicy.blogspot.co.uk/2013/03/working-together-to-reduce-patent.html> accessed 31 August 2020.

[133] 'How We Protect Members' LOT Network <https://lotnet.com/how-we-protect-members/> accessed 31 August 2020.

[134] Promise at Red Hat, 'Statement of Position and Our Promise on Software Patents' <http://www.redhat.com/legal/patent_policy.html> accessed 31 August 2020.

[135] 'Nokia announces patent support to the Linux Kernel' Phys.org (26 May 2005) <https://phys.org/news/2005-05-nokia-patent-linux-kernel.html> accessed 31 August 2020. This pledge has a number of different qualifications, including '[w]ith respect to new functionality introduced into future Linux Kernel releases, Nokia reserves the right to declare that the Patent Statement shall not apply'.

[136] IBM, 'IBM Pledges 500 U.S. Patents to Open Source in Support of Innovation and Open Standards' (11 January 2005) <http://www-03.ibm.com/press/us/en/pressrelease/7473.wss> accessed 31 August 2020.

pledging entities to OIN—including Microsoft in October 2018, which has been followed by an extension of the Linux Definition to add the Microsoft exFAT patents into the Linux System Definition and OIN's pool.

One major unresolved issue is whether a pledge binds a new owner of a patent, an issue of great practical significance given the powerful and accelerating trend for major patent holders to divest some parts of their patent portfolio to patent assertion entities. This issue is also being considered in the context of whether FRAND obligations bind successors in title, as discussed in Chapter 11 .

10.6 Patent Litigations Initiated Against Open Source

Although concerns about the impact of patents against Open Source have been raised for at least thirty years (since GPLv2 identified patents as a concern in its preamble and the 'liberty or death' clause) and although numerous measures, as discussed earlier, have been implemented to address those concerns, actual threats (at least in the form of patent infringement suits filed against Open Source) have been surprisingly rare and generally resolved in a way favourable to the Open Source model. This data is somewhat contrary to the general trend of patent infringement litigation filings, which have shown, in the US, a steady-state of such filings by practicing entities, and a variable—but gradually increasing—trend of filings by NPEs.[137] Global trends also seem to indicate an increasing rate of patent infringement suit filings by NPEs.[138]

Although rare, there have been a few instances of patent infringement litigation filed against software licensed under an Open Source licence. In almost every instance, these litigations have been filed either ancillary to a separate, non-patent, dispute,[139] and in almost all cases, the patent is asserted against a for-profit entity that makes Open Source software part of its overall revenue-producing product profile.[140] In the one case where an actual verdict of patent

[137] See RPX Corporation, 'What 15 Years of U.S. Patent Litigation Data Reveal About the IP Market' *RPX Insights* (25 January 2021) <https://insight.rpxcorp.com/news/65081-what-15-years-of-us-patent-litigation-data-reveal-about-the-ip-market> accessed 20 March 2021.

[138] See Michael Crichton, Gregory Gramenopoulos, Vincenzo Jandoli, et al., 'Global Patent Litigation: Trends, Tools, and Strategies to Enforce Your Patent Rights Globally' *Strafford* (2 June 2020) <http://media.straffordpub.com/products/global-patent-litigation-trends-tools-and-strategies-to-enforce-your-patent-rights-globally-2020-06-02/presentation.pdf> accessed 20 March 2021.

[139] See, e.g., *XimpleWare v Versata,* Case No. 3:13cv5160 (N.D. Cal. 2013) (a copyright infringement action for failure to abide by GPLv2) and *XimpleWare v Versata,* Case No. 5:13cv5161 (N.D. Cal. 2013) (a corresponding patent complaint for patent infringement resulting from the failure to abide by GPLv2). These two suits, as well as other associated suits, were eventually settled. Sylvia Jakob, 'Versata saga settled with prejudice' *ifrOSS News* (19 March 2015) <https://www.ifross.org/?q=en/artikel/versata-saga-settled-prejudice-1> accessed 20 March 2021. Details of the terms of settlement are not public.

[140] See, e.g., *Bedrock Computer Technologies LLC v Softlayers Technology Inc.*, Case No. 6:09-cv-269 (LED) (E.D. Tex. 2009), which involved a patent infringement claim against Google, and others, for features in the Linux kernel. Google initially lost the claim and was assessed damages of U S$5,000,000. Steven Vaughn-Nichols, 'Idiotic Anti-Linux & Google Patent Decision' *ZDNet* (21 April 2011) <https://

infringement was found, and damages were assessed for that infringement, the verdict was rendered in summary fashion via a jury (thus not providing a detailed explanation of how the patent was infringed by the accused Open Source)[141] and was shortly thereafter settled and dismissed without explanation as to the terms of the settlement.

One of the earlier patent litigation assertions against Open Source software was a claim made against Red Hat's distribution of JBoss's Hibernate object-relational mapping tool (licensed under GPLv2) by the patent holder FireStar.[142] No court decision was rendered in that litigation, but upon settlement, Red Hat did make a statement assuring the Open Source community that that settlement was fully conformant with Red Hat's patent obligations under the 'liberty or death' provisions of GPLv2:

> The covered products include all software distributed under Red Hat's brands, as well as upstream predecessor versions. The settlement also protects derivative works of, or combination products using, the covered products from any patent claim based in any respect on the covered products. Essentially, all that have innovated to create, or that will innovate with, software distributed under Red Hat brands are protected, as are Red Hat customers.
>
> 'Red Hat's settlement satisfies the most stringent patent provisions in FOSS licenses, is consistent with the letter and spirit of all versions of the GPL and provides patent safety for developers, distributors and users of FOSS software,' said Richard Fontana, FOSS Licensing and Patent Counsel at Red Hat.[143]

A more recent patent litigation involving Open Source, which demonstrates the complex interplay of patent infringement assertions, the various mechanisms for challenging patents (both administratively and in court, in the United States), and continued controversy about the legitimacy of the mechanism for administratively challenging patents using Inter Partes Review (IPR) can be found in the activities of the patent holder Sound View Innovations (Sound View). Beginning in 2016, Sound View filed a series of patent infringement lawsuits—in

www.zdnet.com/article/idiotic-anti-linux-google-patent-decision/> accessed 20 March 2021. The case was eventually settled, with regard to Google, and dismissed, see Order Vacating Verdict and Dismissing Claims and Counterclaims (18 May 2011) available at <https://docs.justia.com/cases/federal/district-courts/texas/txedce/6:2009cv00269/116887/830> accessed 20 March 2021, although details of that settlement are not public.

[141] Jury Verdict, *Bedrock Computer Technologies LLC v Softlayers Technology Inc.*, Case No. 6:09-cv-269 (LED) (E.D. Tex. 15 April 2011) available at <https://docs.justia.com/cases/federal/district-courts/texas/txedce/6:2009cv00269/116887/746> accessed 20 March 2021.

[142] Paula Rooney, 'FireStar Files Suit Against Red Hat' *CRN* (7 July 2006) <https://www.crn.com/news/applications-os/190300990/firestar-files-suit-against-red-hat.htm> accessed 20 March 2021.

[143] Red Hat, 'Red Hat Puts Patent Issue to Rest' Red Hat Press Release (11 June 2008) <https://www.redhat.com/en/about/press-releases/patent> accessed 20 March 2021.

Delaware,[144] California,[145] and Colorado[146] in the US—accusing a variety of different companies, and a number of different technologies used by those companies, of infringing a portfolio of as many as seven US patents. At least some of these patents were claimed to be infringed by Hadoop data processing software (licensed under the ASF 2.0 Licence) and the JQuery JavaScript library (licensed under the MIT licence).[147] The Sound View patent asserted against Hadoop, US Patent No. 6,125,371 was eventually ruled invalid as the result of an IP filed against it, and that ruling was upheld by the US Court of Appeals for the Federal Circuit.[148]

Although non-commercial Open Source projects have historically avoided patent infringement suits, there is at least one, recent, incident of a direct assertion of patent litigation claims against an Open Source project itself, rather than a commercial entity making a business of distributing Open Source.[149] Rothschild Patent Imaging (RPI), an NPE associated with an inventor with a large number of patents held by many different NPEs, sued the GNOME Foundation's 'Shotwell' feature (licensed under LGPLv2.1) for patent infringement.[150] The patent lawsuit against the GNOME Foundation was ultimately settled with RPI granting a licence to all software—not just Shotwell, or GNOME code—licensed under an OSI-approved licence, without payment of any royalty, fee, or settlement amount, to any patent originating from the same inventor.[151] The scope of that settlement,[152] like the settlement with FireStar by Red Hat, may also have been driven by the 'liberty or death' patent provisions that, like in GPLv2, exist in LGPLv2.1. The Executive Director of the GNOME Foundation, Neil McGovern, expressed complete satisfaction with the ultimate resolution of that patent dispute: 'McGovern said he was 'exceptionally pleased with the outcome … I felt it was incredibly important to send a message to the entire patent assertion industry that basically you don't go

[144] *Sound View Innovations, LLC v. Facebook, Inc.* Case No. 1:16-cv-00116-RGA (D. Del. 2019).

[145] *Sound View Innovations, LLC v Hulu, LLC*, Case No. 2:17-cv-04146-JAK-PLA (C. D. Cal. 2017).

[146] *Sound View Innovations, LLC v Sling TV LLC*, Case No. 1:19-cv-03709-CMA-SKC (D. Col. 2019).

[147] Adam Philipp, 'Sound View Claims Open Source Software Infringes Patents' *AeonLaw* (22 May 2019) <https://aeonlaw.com/blog/2019/05/22/sound-view-claims-open-source-software-infringes-patents/> accessed 21 March 2021.

[148] *Sound View Innovations, LLC v Hulu LLC*, Case: 19-1865 (Fed. Cir. 2 July 2020).

[149] Campbell Kwan, 'GNOME faces 'baseless' patent lawsuit for organising images' *ZDNet* (26 September 2019) <https://www.zdnet.com/article/gnome-faces-baseless-lawsuit-from-patent-troll/> accessed 29 August 2020.

[150] Richard Speed, 'Fairytale for 2019: GNOME to battle a patent troll in court' *The Register* (25 September 2019) <https://www.theregister.com/2019/09/25/gnome_sueball_shotwell/> accessed 20 March 2021. Amanda Brock and Matt Berkowitz, 'GNOME

[151] Amanda Brock and Matt Berkowitz, 'GNOME Settles Litigation, Extends Patent Coverage to all Open Source Initiative Licensing' *The New Stack* (30 July 2020) <https://thenewstack.io/gnome-settles-litigation-extends-patent-coverage-to-all-open-source-initiative-licensing/> accessed 29 August 2020.

[152] The GNOME Foundation did not made the settlement agreement and licence terms public, and all mention of the lawsuit and settlement have been scrubbed from the GNOME Foundation's website. The settlement agreement was nevertheless posted by others, and can be found at the following location: <https://blog.hansenpartnership.com/wp-uploads/2020/09/GNOME_final_agreement_5-20_with_schedules.pdf> accessed 20 June 2022.

after open source projects. It won't end well for you.'[153] Whether the patent litigation against the GNOME Foundation represents an anomaly, or the start of a trend of NPEs asserting patents directly against projects themselves, remains to be seen.

10.7 Conclusions

The Open Source community attitude to patents has gone from raising the issue—rejecting software patents on principle—to implementing sophisticated mechanisms for dealing with them, both on a structural basis (in Open Source licences) and in public and private initiatives. Looking back at the initial objective of exploring the relationship between patents and Open Source, it can be seen that there are several areas of friction, creating risk and uncertainty. However, the different mechanisms mentioned that aim to reduce these issues are far from completing the task. What more can be done?

10.7.1 If you can't beat them … should you join them?

One view to take is that as the software patent system seems to be here to stay (in one form or another), the Open Source community should become a participant in the system if it wishes to protect itself from the threats of patent thickets, patent lawsuits, International Trade Commission (ITC) proceedings, patent-encumbered standards, and high awards in the event of infringement findings.[154] This means not only applying for patents and using them as to support defensive countermeasures (something in which Open Invention Network is actively involved) or aggressive measures, potentially creating patent pools for Open Source environments, but also providing Open Source technologies and ideas as searchable prior art and eventually taking a patent licence over Open Source technologies in terms that benefit the whole community, and which comply with copyleft licensing terms.[155]

However, this comprehensive approach is difficult in economic terms, considering the modest financial status of the great majority of Open Source projects

[153] Tim Anderson, '"This was bigger than GNOME and bigger than just this case." GNOME Foundation exec director talks patent trolls and much, much more' *The Register* (23 October 2020) <https://www.theregister.com/2020/10/23/this_was_bigger_then_gnome/> accessed 20 March 2021. After the settlement was made, a separate challenge to the Rothschild patent involved in the GNOME Foundation patent lawsuit was made in the USPTO, resulting in every claim in that patent being cancelled, thus reverting the subject matter of the Rothschild patent asserted against GNOME to the public domain. See OSI Staff, 'GNOME patent troll stripped of patent rights', *Voices of Open Source* (28 April 2022) <https://blog.opensource.org/gnome-patent-troll-stripped-of-patent-rights/> accessed 20 June 2022.

[154] See, e.g., the arguments of Schultz and Urban,'Protecting Open Innovation', note 51.

[155] Two representative examples would be the settlements negotiated to resolve the FireStar against Red Hat, and the Rothschild assertion against GNOME, both of which were reported to have been settled under community-beneficial licence terms (see section 10.6 earlier).

(commercial or not), because they require substantial industry backing, such as the way in which the OIN and its various initiatives have the financial backing of significant market players such as Philips, NEC, Sony, IBM, Red Hat, Google, Toyota, and SUSE.

10.7.2 Patent reform

More recently, there have been a number of proposals for patent reform, the idea being that in the context of these conflicts, rather than forcing Open Source development to change and adapt its ways and methods (which have been proven to provide significant innovation and contribution to the 'Progress of Science and useful Arts')[156] to a legal framework that is unaligned with the functioning of the Open Source model, that instead the legal system itself that should be improved. Indeed, there are those that argue that the patent system in general has not led to greater innovation, especially in the field of software, as much as constituting a block on innovation and progress.[157]

Some writers have suggested significantly modifying the patent system, reducing the strength of patent protection, if not getting rid of patents altogether (at least for software), a view taken not only by the FSF[158] and the Foundation for a Free Information Infrastructure[159] but also some leading academics in the field.[160] Proposals include expressly eliminating or limiting software as patentable subject matter, tailoring the length of patent protection to software (to a period of much less than the current twenty years from first filing), or awarding patents only when

[156] US Const, Art I, s 8, cl 8, known as the patent and copyright clause.

[157] James Bessen and Michael Meurer, *Patent Failure* (Princeton, NJ: Princeton University Press, 2008), have found evidence that patents can actually harm innovation. Eric von Hippel concluded that 'empirical data seem to suggest that the patent grant has little value to innovators in most fields' in Eric von Hippel, *Sources of Innovation* (Oxford: Oxford University Press, 1988) available online at <http://web.mit.edu/evhippel/www/sources.htm> accessed 19 March 2021. In *The Wealth of Networks* (New Haven, CT: Yale University Press, 2006), Yochai Benkler suggests that patents may result in a drop in productivity. In Josh Lerner, 'Patent Protection and Innovation over 150 Years' (Nat'l Bureau of Economic Research, Working Paper No 8977, 2002), the author noted that strengthening available patent protection tended to yield less patenting of new innovations by domestic inventors, which may correlate with reduced rates of technological innovation.

[158] Richard M Stallman, 'Software Patents—Obstacles to Software Development' in *Free Software, Free Society: The Selected Essays of Richard M. Stallman* , see note 10.

[159] FFII <http://www.ffii.org>; and Stop Software Patents, 'Petition to stop software patents in Europe' <https://www.devroom.io/2010/01/19/sign-the-petition-stop-eu-software-patents/> both accessed 19 March 2021.

[160] Boldrin and Levine, *Against Intellectual Monopoly,* see note 52, conclude that 'a system that at one time served to limit the power of royalty to reward favoured individuals with monopolies has become with the passage of time a system that serves primarily to encourage failing monopolists to inhibit competition by blocking innovation' (at 20). See also James Bessen and Michael Meurer, 'The private costs of patent litigation' (Boston University School of Law Working Paper Series, Law and Economics, Working Paper No 07–08, online at <http://dx.doi.org/10.2139/ssrn.983736> accessed 19 March 2021), the authors conclude: 'In the worst case, the net effect of patents today may be to reduce the profits of public firms and to possibly impose disincentives on innovative activity as well.'

strictly needed on economic grounds (although the latter would be difficult given that most patent offices are ill-equipped to evaluate economic data).

Along similar lines, other more moderate changes have been proposed, to limit the effect of patents in the context of software. At a conference on Patent Reform at Santa Clara Law School,[161] Professor Mark Lemley, one of the leading advocates of patent reform in the US, suggested that the interpretation of US patent law should be tightened up, to prevent software patents from being drafted in general functional terms (thus prohibiting any implementation of the functional idea, creating an overbroad patent), and limit enforceable claims to the actual algorithms disclosed by the patentees and their equivalents. This rule is something is argued that the courts in the US should be doing under the Patent Act of 1952,[162] increasing disclosure obligations for software related patents and details of computer implemented functional claims, obliging applicants, for example, to use diagrams, flowcharts, or pseudocodes along with a clear description of the invention in natural language, and reducing the abstract nature of claims. This idea is also of some interest to the European Patent Convention regime, which generally allows functional claims but only to the extent that any more precise definition would reduce the scope of the invention (which is in fact the very purpose of ruling out functional claims).[163] The EPO Guidelines develop this, prohibiting attempts to define an invention purely in terms of the result to be achieved (thus claiming the underlying technical problem), particularly if a claim is formulated in such a way as to embrace other means, or all means, of performing the function.[164]

Another suggested idea is not to attack the upstream source of the problem, the patentability of software, which is proving to be fairly immutable,[165] but to limit the effect or enforceability of software patents on the market, reducing the liability risk for Open Source projects and users. One proposal is to legislate a 'safe harbour' from patent claims for software that runs on 'general purpose machines' (PCs and servers, terminal and mobile devices such as smart phones, routers, and set-top boxes, and so forth).[166] This may seem rather conservative, for example it would not apply to specifically programed hardware devices, and doesn't really deal with existing patents (unless the effect would be retroactive with regard to

[161] Santa Clara Law, 'Solutions to the Software Patent Problem' (16 November 2012) <https://law.scu.edu/hightech/2012-solutions-to-the-software-patent-problem/> accessed 19 March 2021.

[162] US Patent Act—35 USC, Article 112. See Mark Lemley and Julie Cohen, 'Patent Scope and Innovation in the Software Industry' (2001) 89 *California Law Review* 1. See also Ballardini, 'The Software Patent Thicket', see note 32.

[163] Article 83 EPC. See *Synergestic herbicides/CIBA GEIGY* T68/85, and subsequent cases.

[164] EPO Guidelines, C-III, 4.10 and 6.5.

[165] See the summary of the debate around software patents that occurred in the EU in Free Software Foundation-Europe, 'Software Patents in Europe', *FSFE Activities* <https://fsfe.org/activities/swpat/swpat.en.html> accessed 20 June 2022.

[166] Richard Stallman, 'Let's Limit the Effect of Software Patents, Since We Can't Eliminate Them' *Wired* (1 November 2012) <http://www.wired.com/opinion/2012/11/richard-stallman-software-patents> accessed 19 March 2021.

issued patents). Another suggested approach is to focus on interoperability and standards and only allow software patents to be enforced against implementations of standards where the patents had been previously declared during the standard setting process. 'All other software contexts should become off-limits for patent enforcement.'[167]

In the absence of any reform—a prospect that the Open Source community has advocated for more than twenty years but has never come close to fruition—Open Source projects must resort to classic defence strategies to deal with patent risks: obtaining a licence, proving non-infringement, proving invalidity due to lack of novelty, obviousness/lack of inventive step, or inventiveness (or requesting review, on the same bases), getting legal opinion support for invalidity or non-infringement (to reduce claims of wilful infringement), looking for other grounds for non-enforceability such as expiry, and eventually, of course, the technical solution of designing around the patent.[168]

[167] Simon Phipps, 'Stop patent mischief by curbing patent enforcement' *Infoworld* (9 November 2012) <http://www.infoworld.com/d/open-source-software/stop-patent-mischief-curbing-patent-enforcement-206658> accessed 19 March 2021.

[168] See Richard Fontana et al, 'A Legal Issues Primer for Open Source and Free Software Projects' *Software Freedom Law Center* (2008) <http://www.softwarefreedom.org/resources/2008/foss-primer.html> accessed 19 March 2021.

11

Open Source Software in Standard Setting

The Role of Intellectual Property Right Regimes

Knut Blind, Mirko Böhm, and Nikolaus Thumm

11.1 Introduction

In the communication 'Setting out the EU Approach to Standard Essential Patents' the European Commission[1] announced in 2017 that it would analyse complementary possibilities for interaction and differences between Open Source software and standardisation processes, and recommend solutions for the smooth cooperation between standardisation and Open Source. Prior to this, the interface between Open Source software and standardisation had been only marginally touched both by researchers and practitioners in standardisation bodies or Open Source communities.

[1] European Commission, 'Communication from the Commission to the Institutions on Setting out the EU approach to Standard Essential Patents' <https://ec.europa.eu/docsroom/documents/26583> accessed 11 March 2021

Knut Blind, Mirko Böhm, and Nikolaus Thumm, *Open Source Software in Standard Setting* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Knut Blind, Mirko Böhm, and Nikolaus Thumm 2022. DOI: 10.1093/oso/9780198862345.003.0011

This chapter builds on results of the European Commission report by Blind and Böhm[2] on the interrelation between standardisation and Open Source software, which refers in particular to the interaction between Open Source software and FRAND (fair, reasonable and non-discriminatory) patent licences in standardisation. FRAND commitments aim to ensure that essential technology protected by intellectual property rights (IP) included in a standard is made available to users of that standard on fair, reasonable, and non-discriminatory terms. FRAND commitments aim to prevent IP holders from refusing to license patents and from charging licensees excessive fees (unfair or unreasonable) for standard implemented patented technologies.

The intersection of Open Source software and FRAND licensing and its integration into the process of standard setting is considered throughout this chapter. It does not consider other FRAND licensing related issues, nor does it analyse specific FRAND court decisions. Comino, Manenti, and Thumm[3] provide an overview of the wider FRAND-related economic issues, and Baron and Pentheroudakis[4] provide a comprehensive analysis of the most important FRAND court cases.

The complex interface of Open Source and standardisation processes is analysed with a specific focus on the role of IP including FRAND licensing. Standards and Open Source development are both processes widely adopted in the information and communications technology (ICT) industry to develop innovative technologies and drive their adoption in the market. Innovators and policy makers often assume that a closer collaboration between standards and Open Source development would be mutually beneficial. The interaction between the two is however not yet fully understood, in particular with regards to how the IP regimes applied by these organisations influence their ability and motivation to cooperate. Most Standards Development Organisations (SDOs) use FRAND licensing terms, while widely used Open Source licences like the General Public Licence (GPL) are largely incompatible with these terms.[5]

[2] Knut Blind, Mirko Böhm, and Nikolaus Thumm (ed), *The Relationship between Open Source Software and Standard Setting* (Brussels: European Commission 2019).

[3] Stefano Comino, Fabio Manenti, and Nikolaus Thumm, 'The Role of Patents in Information and Communication Technologies. A Survey of the Literature' (2019) 33(2) *Journal of Economic Surveys* 404–30.

[4] Justus Baron, Chryssoula Pentheroudakis, and Nikolaus Thumm, (ed), *Licensing Terms of Standard Essential Patents, A Comprehensive Analysis of Cases* (Brussels: European Commission 2017).

[5] Catharina Maracke, 'Free and Open Source Software and FRAND-based Patent Licenses: How to Mediate between Standard Essential Patent and Free and Open Source Software' (2019) 22(3–4) *The Journal of World Intellectual Property* 78–102.

11.2 Results from the Literature

Relevant studies are divided into three categories, according to Lundell and Gamalielsson,[6] and Clark,[7] without immediately taking into account the tension between Open Source software licences and the FRAND regime regarding patents:

- The first cases begin were standardisation projects within formal or informal standardisation bodies but are eventually implemented as Open Source projects.
- The second option is the initial implementation of software via Open Source projects, followed by a standardisation process.
- The third and last option is the parallel development of standards and their implementation as Open Source .

11.2.1 Standards implemented as Open Source software ('standard first')

Examples of the implementation of standards via Open Source software are discussed but are mainly developed in SDOs under a RF licensing scheme, for example the Organization for the Advancement of Structured Information Standards (OASIS) and the World Wide Web Consortium (W3C). According to Phipps,[8] these SDOs are characterised by an implementation-oriented rather than requirement-oriented standardisation approach. There are also some standards published by the SDOs using the FRAND regime. However, there are no declarations of Standard Essential Patents (SEPs) referring to these standards.

Nevertheless, due to the general contradiction between the FRAND regime and some Open Source licences, there is still a latent fear of conflicts with potential SEP holders, and the popular GPL licence is not compatible with FRAND.

[6] Björn Lundell and Jonas Gamalielsson 'On the Potential for Improved Standardisation through the Use of Open Source Working Methods in Various Standardisation Organisations: How Can Open Source Projects Contribute to the Development of IT Standards' in Kai Jakobs and Knut Blind (eds), *Digitalisation: Challenge and Opportunity for Standardisation: Proceedings of the 22nd EURAS Annual Standardisation Conference* (Mainz: Verlag Mainz, 2017) 137–55.

[7] Jamie Clark, 'Konvergenz, Zusammenarbeit und intelligentes Einkaufen in offenen Standards und Open Source (Diashow)' (ITU/NGMN Joint Workshop on OS, San Diego 2016) <https://www.slideshare.net/JamieClark1/oasis-at-itungmn-convergence-collaboration-and-smart-shopping-in-open standards-and-Open Source-81474699> accessed 7 March 2021.

[8] Simon Phipps 'Open Source and FRAND: Why Legal Issues Are the False Lens to Understand the Open Source and FRAND issue' OpenForum Europe (2019) <http://www.openforumeurope.org/wp-content/uploads/2019/03/OFA_-_Opinion_Paper_-_Simon_Phipps_-_OSS_and_FRAND.pdf> accessed 7 March 2021.

There is some assertion that Open Source software licences, such as the MIT or BSD licences, may be compatible with FRAND.[9] However, there is no general consensus on this conclusion, as others argue[10] equally that these are only complementary licences and are not compatible.[11] Furthermore, the idea of incompatibility of specific licence systems with a FRAND regime is supported by a significant percentage of Open Source programers.[12]

The concerns of the Open Source software communities regarding the lack of clarity of FRAND licensing conditions is becoming more important due to the increasing relevance of the successful implementation of standards for quality and success. These may also, as with other patent concerns, be countered in a number of defensive IP structures as discussed in Chapter 10.

11.2.2 Open Source code as input into a standard ('software implementation first')

The second category, according to Lundell and Gamalielson,[13] is characterised by the initial implementation of software, which ultimately leads to technical specifications of standards, which Phipps[14] also calls 'implementation-led standardisation'. In this scenario, a software implementation precedes the development and approval of the technical specifications of a standard published by either a formal and/or informal SDO. According to Li,[15] it is generally more complicated for SDOs to use Open Source working practices to develop standards.

In addition to the inclusion of software code in the technical specifications of standards, the functions of the code can also be transferred to a standard. Li[16] makes a distinction here between the different licences applicable to the Open Source code. If the licence does not contain a patent clause, the patent issue is still important under the policy of the relevant SDO, possibly subject to a FRAND obligation (under Li 2017). However, if a licence contains a patent clause, the patent right in that licence is granted on an RF basis and leaves open the question of

[9] David Kappos and Miling Harrington, 'The Truth About OSS-FRAND: By All Indications, Compatible Models in Standard Settings' (2019) XX The *Columbia Science and Technology Law Review*.

[10] European Commission, 'Implementation of FRAND standards in Open Source: Business as usual or Mission impossible?', see note 3.

[11] Phipps 'Open Source and FRAND', see note 8.

[12] Rudi Bekkers and Andrew Updegrove, *A Study on Intellectual Property Rights Policy and Practice of a Representative Group of Standards Setting Organizations Worldwide* (Washington: National Research Council, 2012)

[13] Lundell and Gamalielsson 'On the Potential for Improved Standardisation', see note 6.

[14] Phipps 'Open Source and FRAND', see note 8.

[15] Jingze Li 'Licensing tensions in the context of the inclusion of Open Source in the context of formal standard setting—The case of Apache V.2 in ETSI as a beginning' *ITU Kaleidoscope* (2017) <https://ieeexplore.ieee.org/document/8246986> accessed 7 March 2021.

[16] Li 'Licensing tensions in the context of the inclusion of Open Source in the context of formal standard setting', see note 15.

whether SDOs can require patent holders who contribute patents to the standard to license these under FRAND licensing, where there is already an Open Source licence with RF patent licences, and how any inherent conflict between the two would be resolved. Li[17] states that this is not yet established in the current IP regulations of the SDOs.

Since most SDOs have not incorporated specific rules for licensing Open Source code into the specifications of standards, Li[18] concludes that the Open Source licensing terms are the 'only clearly applicable rule'. Consequently, the granting of RF licences for the use of the code in embedded standard technologies should be applied.

However, such rules could be a strong disincentive for at least some innovators holding patents, as this would remove the possibility of charging royalties on SEPs, which might be one of the main incentives for many innovators to contribute to standardisation.[19] In the survey by Blind and colleagues,[20] patent-owning companies rated the relevance of the freedom of action achieved by a standard as a much higher incentive than any associated royalty payment.

In addition to the conflict between the licensing conditions for Open Source and patents, there is a systemic conflict in licensing software under the Open Source licensing conditions and patents under FRAND. Open Source licences follow a cascade effect that restricts implementers in other areas not covered by FRAND.[21] Although patents are used free of charge, licences generally contain a 'patent retaliation clause' that prevents recipients from litigating against the work, including the patented contribution by terminating the patent right. This is intended to prevent implementers from filing a lawful lawsuit if they find that their patents contained in the same work have been infringed. However, the current IPR regimes of the SDOs guarantee patent holders this possibility.

In summary, the current frameworks utilised by both formal SDOs and informal consortia seem to allow the integration of Open Source into their standard development process and standards. SDOs, such as W3C and OASIS, have more of an RF culture with regard to patents and consequently have a rather limited number or no SEPs at all, and are pioneers in launching proactive initiatives to include Open Source in their standards. Despite the challenges for SDOs using FRAND, the exclusion of Open Source code from the specifications of the standards is not a

[17] Li 'Licensing tensions in the context of the inclusion of Open Source in the context of formal standard setting', see note 15.

[18] Li 'Licensing tensions in the context of the inclusion of Open Source in the context of formal standard setting', see note 15.

[19] For example, Josh Lerner and Jean Tirol 'Standard-Essential Patents' (2015) 123(3) *Journal of Political Economy* 547–86

[20] Knut Blind, Rainer Bierhals, Eric Iversen, et al, *Study on the Interaction between Standardization and Intellectual Property* (Karlsruhe: Fraunhofer Institute for Systems and Innovation Research, 2002)

[21] Li 'Licensing tensions in the context of the inclusion of Open Source in the context of formal standard setting', see note 15.

sustainable strategy, as the available code base is already large and widely adopted which continues to grow.

In addition, some Open Source communities claim to set *de facto* standards, which calls into question both formal SDOs and informal consortia.[22]

Finally, both the increasing competition between SDOs and consortia and the additional competition from the Open Source communities as additional standard setters are likely to increase the pressure to cooperate with the latter. Industry standards can be developed through competition between communities, with no formal specification at all.

11.2.3 Open Source and standardisation in parallel ('standard and implementation of the standard in parallel')

In the two previous categories, a clear distinction was made between the starting point of the process and the transfer to the other area. Category three, however, represents the interaction between the development of technical specifications of a standard together with the development of one (or more) implementation(s) of technical specifications of a standard in Open Source software.

Lundell and Gamalielsson,[23] who further develop Gamalielsson and colleagues,[24] analyse the bi-directional influences between the Open Source project Drupal, which is distributed under the copyleft GPLv2.0 licence, and the development of the RDFa standard for data exchange on the web at the W3C. Support for RDFa 1.0 was achieved in Drupal through its first implementation in the core of Drupal 7 (RDFa is implemented in a separate module in Drupal).[25]

The summary of the findings in connection with the third category of parallel developments in Open Source and standardisation confirms the observations made in connection with the first and second categories. In the early days of the Internet, the International Engineering Task Force (IETF), as a consortium driven by individual members, like Open Source projects, was involved in the development of an email format in parallel with Open Source projects. The few cases of close interaction between Open Source and standardisation are mainly concentrated in consortia with a strict RF and rather patent-incompatible licensing policy, that is W3C or OASIS. It has already been noted that they are in a better position to integrate input from Open Source projects, as opposed to the formal SDOs that use the FRAND structure.

[22] Andrew Updegrove 'Licensing standards that contain code: Heads or tails?' ConsortiumInfo.org (2015), <http://www.consortiuminfo.org/standardsblog/articles/licensing-standards-include-code-heads-or-tails> accessed 7 March 2021.

[23] Lundell and Gamalielsson 'On the Potential for Improved Standardisation', see note 6.

[24] Jonas Gamalielsson, Björn Lundell, Jonas Feist et al, 'On Organizational Influences in Software Standards and Their Open Source Implementation' (2015) 67 *Information and Software Technology* 30–43.

[25] Lundell and Gamalielsson 'On the Potential for Improved Standardisation', see note 6.

The recursive integration of inputs from standardisation or Open Source can lead to a 'virtuous circle' of standards of higher quality and wider dissemination. In contrast the challenges for the FRAND-based SDOs and consortia will also create difficulties for parallel innovation developments. In the long term, higher quality standards due to Open Source inputs and their wider dissemination through Open Source adoption will further increase the pressure on formal SDOs and informal consortia under the FRAND regime.

The limited focus of SDOs on copyright in general and on software or Open Source in particular has a number of economic implications. First, the few SDOs or consortia that deal explicitly with software are able to develop a stronger profile in the standardisation of topics based on software alone or on the combination of software and hardware. Here, the actors with a need for standardisation obviously decide according to the perceived competencies of SDOs, including governance in connection with software. Secondly, the FRAND regime, which is relevant for the licensing of SEPs established in traditional SDOs, in other words members of the International Organization for Standardization (ISO), and therefore guided by ISO/ International Electrotechnical Commission (IEC)/International Telecommunication Union (ITU) policies on IP,[26] does not necessarily attract contributors to Open Source used to more RF-dominated licensing systems. Therefore, separation or division of labour is likely to continue in the future, despite the considerable efforts of ETSI, in particular, to find solutions for the coexistence of FRAND and Open Source licences, as expressed by some of its members in the Fair Standards Alliance.[27] Secondly, the rather strict RF-based policies of OASIS and W3C,[28] which follow the Open Source licensing regime, facilitate the implementation of their standards. Thirdly, the IP policy of SDOs in relation to software is linked to their business models, especially those that do not make their services freely available. It is more difficult to sell standards under a freely licensed regime that integrates Open Source into standards. However, this area of tension has not yet been addressed in the rules of the SDOs and consortia.

11.2.4 Summary of the literature

The background, as presented in the literature review, leads to very general conclusions regarding the growing importance of standard-setting processes and their use of IP, not only patents but also copyright to software. The general assessment

[26] Justus Baron, Jorge Contreras, Martin Husovec, et al, and Nikolaus Thumm (ed), *Making the Rules; The Governance of Standard Development Organizations and their Policies on Intellectual Property Rights* (Brussels: European Commission 2019).

[27] Fair Standards Alliance 'The Importance of Maintaining the Open Source Software Value Proposition' (Brussels: Fair Standards Alliance, 2017) <http://www.fair-standards.org/wp-content/uploads/2016/08/FSA-Maintaining-The-Open source-Software-Value-Proposition.pdf> accessed 7 March 2021

[28] Baron, Contreras, Husovec, et al, and Thumm (ed), *Making the Rules*, see note 26.

is that standardisation plays a multidimensional role; it mediates between science- and technology-driven research and innovation and demand-oriented innovation policy,[29] which is framed by various regulatory regimes. Patents and software, including Open Source, are the main IP rights used as input for ICT standardisation and are also relevant for the accessibility of ICT standardisation results. Therefore, the use of IP in standardisation processes adds another dimension.

In general, standards are developed by a number of different actors in a voluntary, consensus-based process. In view of the associated increasing diversity of interests of the actors involved in standard-setting, governance in standard-setting determines the success of SDOs in terms of integrating the various interests, which is also called for by the European Commission. Effective rule-setting and governance of SDOs are critical to the successful development and, ultimately, implementation of standards. IP policies developed by SDOs will have to take into account not only specific rules and procedures for FRAND licensing relevant to patents but even more so the treatment of Open Source .

11.3 Insights from Case Studies and Stakeholder Consultation

The recent study by Blind and Böhm[30] for the European Commission comprises a detailed empirical investigation of the interaction between standard development organisations and Open Source communities. It is based on twenty case studies, a survey of stakeholders (more than 300 respondents) from SDOs and Open Source and an expert workshop. The case study analysis revealed different views on the nature of possible collaboration between SDOs and the wider Open Source community. The most commonly used thinking model for 'working with the Open Source community' within SDOs is the expectation that SDOs will develop specifications into standards, and the Open Source community will then implement them. This approach is based on the assumption that a specification is initially created as part of a standards development process, and the creation of a concrete, compliant product is left to the implementers competing in the market. As discussed, this specification-oriented approach to standardisation is only used in a minority of Open Source instances.

The cases and literature revealed that two-thirds of these can be considered as highly innovative, large-scale collaborations that have found broad or sometimes worldwide acceptance, for example Java, Linux, and PDF. Almost one-third had a significant impact on a specific market segment. In the most recent cases, no impact has been realised as of yet, although they are considered innovative by the

[29] OECD, *Demand-Side Innovation Policy* (OECD 2011).

[30] Blind, Böhm, and Thumm (ed), *The Relationship between Open Source Software and Standard Setting*, see note 2.

participants. About half of the cases achieved market-wide relevance across several industries as basic technologies or by promoting business-critical infrastructure.

There is no clear definition of industrial sectors and sub-sectors concerned, as technologies dealt with in the case studies have many uses. As a general cross-industry trend, computer and telecommunications systems are becoming fundamental technologies for various products and business processes.

The choice of an early, parallel, or late approach to standardisation neither limits the chances of success of a project nor is a specific approach as a prerequisite for a successful standard. However, the incubation of new technologies and functions today is more often done through joint implementations or reference implementations under Open Source licences. Most of the innovations to which the cases refer are presented to the SDOs as soon as proven implementations exist and are generally available. Some participants in the case studies stressed that they do not see the development of standards as a means of creating real innovation but as a means of building industry consensus on available technologies to enable economies of scale. In general, models of governance and cooperation must be seen by the relevant actors as appropriate to motivate them to participate, since the most widespread technologies are also those that attract a large number of participants in their development.

In summary, the case studies have developed a partial focus on the networks and telecommunications sub-sector. This was expected since the interaction between standards and development is naturally located at software–hardware interfaces and the telecommunications industry has a more established history of standards practices. The cases show that there are numerous successful collaborations between standards and Open Source development and that they have developed mature, well-established governance, such as ECMA TC39 or ISO JTC1.[31] Both Open Source and SDO procedures are suitable for the development of technical solutions on both a small and large scale. Those of the observed collaborations that introduced explicit patent licensing systems opted for RF *ex ante* licensing with symmetrical terms between contributors and between domestic and foreign licences. SDOs are usually not the driving force for technical developments. More often, Open Source communities hatch new technical solutions until they become candidates for standardisation and market penetration. The Open Source umbrella organisations and foundations (see Chapter 18) increasingly offer functions such as platforms for collaboration and consensus building, which have traditionally been provided by SDOs.

In the study by Blind and Böhm,[32] stakeholders were also asked about their assessment of the interaction between Open Source and standardisation in terms of

[31] Baron, Contreras, Husovec, et al, and Thumm (ed), *Making the Rules,* see note 26.

[32] Blind and Böhm, and Thumm (ed), *The Relationship between Open Source Software and Standard Setting*, see note 2.

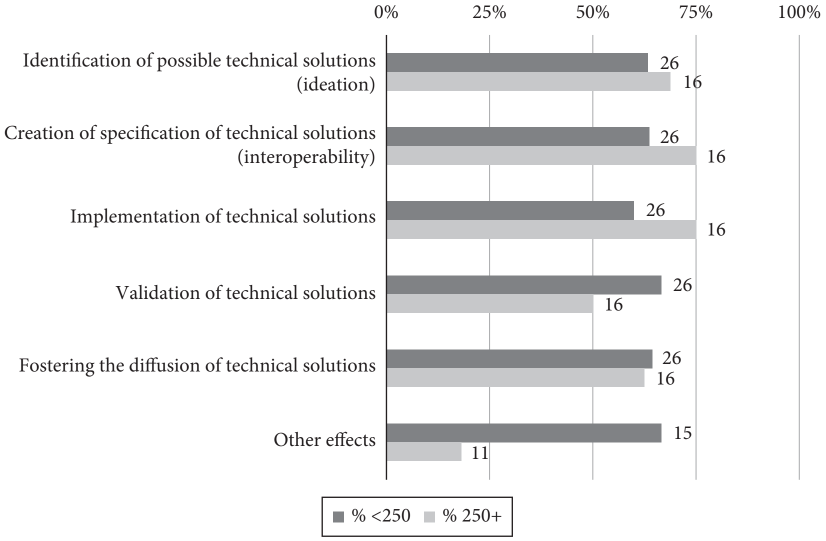


Figure 11.1 Positive impact of interconnection of Open Source and standardisation on efficiency and results of Open Source—SMO vs LO

Source: Blind and Böhm 2019, p. 156.

Blind and Böhm, and Thumm (ed), *The Relationship between Open Source Software and Standard Setting*, see note 2.

efficiency and results. The majority of participants saw a positive effect of this link. In particular, around 70 per cent of respondents saw a positive impact on the development of specifications for technical solutions contributing to interoperability and on the implementation of technical solutions. The benefit of standardisation lies less in the idea of new technical solutions and more in their validation and ultimately in their dissemination, since only about 60 per cent of those surveyed expect positive effects in this area. Negative effects of the interaction on standardisation are generally not to be expected. The distinction between small and large organisations shows that the former tend to expect a positive impact on the identification of possible technical solutions, that is the idea finding, and the drafting of specifications of technical solutions, that is interoperability, while the latter see the benefits particularly in the implementation of technical solutions.

Looking at the effects of interaction on Open Source, we observe an even higher proportion of respondents perceiving positive effects. More than 75 per cent expect a positive impact on the development of specifications for technical solutions, particularly in the context of interoperability, and on the implementation of technical solutions. While about 70 per cent see positive effects on Open Source both for the identification of possible technical solutions and their dissemination, less than 60 per cent of the respondents expect positive impulses for the validation of technical solutions. Again, no negative effects were found. The distinction between small and large organisations, in contrast to the expected impact on standardisation, shows that the former expect rather positive effects on the validation and dissemination of technical solutions while larger organisations again see the advantages in the implementation of technical solutions, but also in the identification of possible technical solutions in the field of Open Source.

Comparing all assessments of networking in terms of efficiency and results, it becomes clear that smaller organisations perceive the knowledge flow from Open Source to SDOs in such a way that the latter receive new ideas as input for technical solutions. Larger organisations see advantages for SDOs of Open Source in implementing technical solutions. In contrast, smaller organisations experience positive effects of standardisation on Open Source on the validation and dissemination of technical solutions. There are a complementary impacts explained by the size of the organisations.

11.4 Compatibility of Intellectual Property Regimes in Standards Development Organisations and Open Source Software

The limited research that deals explicitly with the interaction of SEP licences and Open Source focuses primarily on the legal compatibility of Open Source licences with the FRAND licensing of SEP. This is an important dimension of interaction since any directly contradictory condition in a given combination of Open Source and FRAND licence would prohibit a combination of the two works in a product. However, Blind and Böhm,[33] as well as Maracke[34] and Phipps,[35] point out that answering the question of legal compatibility is not a sufficient precondition for possible cooperation between SDOs and Open Source.

Blind and Böhm[36] find that the question of legal incompatibility can only be assessed in relation to a specific contractual situation and the individual conditions applied in the specific Open Source and FRAND licences. Legal compatibility checks only produce useful results in a specific licensing relationship with a specific Open Source licence in conjunction with specific FRAND terms. Even if a case does not show any incompatibilities, this only means that cooperation is legally possible, not that participants from SDOs and Open Source software would be willing to participate and contribute to a common result. The legal compatibility of the licensing conditions is a necessary condition, but not sufficient to

[33] Blind and Böhm, and Thumm (ed), *The Relationship between Open Source Software and Standard Setting*, see note 2.

[34] Maracke, 'Free and Open Source Software and FRAND-based Patent Licenses', see note 5.

[35] Phipps 'Open Source and FRAND', see note 8.

[36] Blind and Böhm, and Thumm (ed), *The Relationship between Open Source Software and Standard Setting*, see note 2.

establish a successful cooperation between the SDOs and the Open Source software communities.

IP regimes serve different purposes in SDOs and in Open Source communities. Open Source licences mirror and follow collaboration models and represent how participants imagine the jointly created products to be used, resulting in a classification into strong-copyleft, weak-copyleft, and permissive Open Source licences. Governance within the Open Source communities initially developed as a model of cooperation and is then reinforced by choice of one or more licences.

In contrast, the IP frameworks of SDOs regulate how participants engage and how conflicts are resolved. Special attention is paid to how participants can later withdraw from pre-competitive collaboration in SDOs and compete again in products that implement the developed standard. This rationale is alien to the Open Source communities, as they do not intend to re-enter the competitive arena once a functional area is covered by an industry standard Open Source implementation. This contradiction could predict the idea that an Open Source community is creating reference implementations to a standard alongside other competing implementations. In the same context, Open Source communities see no benefit in participating in the development of standards only to facilitate alternative or competing implementations.

The investigation by Blind and Böhm[37] does not provide evidence that the limited cooperation between standards and Open Source development is caused by the uncertainty about the legal compatibility between SDOs and Open Source in the IP regimes. Most of the Open Source projects observed in this study use licences with reciprocal conditions or explicit patent grants and interacted productively with the SDOs relevant for their market segment. Some SDOs have responded to Open Source-related market changes by introducing flexible, or toll-free IP regimes such as W3C and OASIS, and adopting Open Source inspired methods of collaboration. The study by Blind and Böhm[38] does not find a need to reconcile SDOs and Open Source IP policies. With the exception of the telecommunications subsector, there seems to be no conflict between SDOs' IP policies and the policy of Open Source IPs.

It seems that in practice participants adapt to the methods of cooperation and IP policies applied by the communities with which they cooperate, compromising between the contribution of their own IP and access to the overall contributions of the participants. For activities at the interface between standards and Open Source development, this usually means the introduction of a RF patent licensing policy for all examined cases except ETSI-NFV and OpenAirInterface, which actively

[37] Blind and Böhm, and Thumm (ed), *The Relationship between Open Source Software and Standard Setting*, see note 2.

[38] Blind and Böhm, and Thumm (ed), *The Relationship between Open Source Software and Standard Setting*, see note 2.

anticipate the inclusion of FRAND-licensed SEPs in the developed standards. This expectation of RF licensing is considered acceptable and is not an obstacle to co-operation or the development of relevant standards.

In contrast to the distinction between licence-free and FRAND used in standardisation, several licensing models have been developed for Open Source, as discussed in Chapter 3. Looking at the most common regimes the Apache Licence 2.0, the MIT Licence, and the GPLv2.0 are the three most common in the study by Blind and Böhm,[39] followed by the GNU 3 and the BSD Licence 2.0. This ranking largely corresponds to the already publicly available data, which confirms both the validity of the selected cases and the representativeness of the sample. Less common are the GNU LGPL 2.1, the Simplified BSD Licence, the GNU LGPL 3.0, and the Eclipse Public Licence.

In addition to the significant differences in the general attractiveness of the various Open Source licensing models, there are discrepancies between larger and smaller organisations. The latter prefer both the GNU General Public Licence 3.0 and the GNU LGPL 3.0 and the former are inclined towards the permissive MIT and BSD licences. In the case studies, however, it can be observed that many licence selections are made in the early phase of an Open Source project and are never changed. This supports the assertion that licence choice in communities follows the collaborative model that contributors seek, and that newer projects with more corporate contributors more often choose licences with explicit patent clauses, as in the Apache 2 or GPL 3 licences, as opposed to implicit or missing patent licence terms.

Overall, a framework for patent licensing has been established, either through the use of Open Source licences, which involve the granting of patents owned by the participants or by requiring a declaration by the SEP or a commitment by the participants to patent licensing. Although host organisations allow an option for FRAND-based patent licensing in several cases, almost all cases have opted for a RF patent licensing policy. This is either because patents whose claims cover standardised functionality have expired, as in the case of C++, or because the working group aims to make the standard freely available, with the policy of no fees being implemented through a contributor licence agreement.

In case of conflict, the strict separation between Open Source and FRAND licences is still the preferred option, followed by negotiations to find solutions based on the experiences reported in specific cases. If no solutions are found, small organisations, in particular, will withdraw from standardisation. Another possibility is the use of pure copyright licences, which explicitly exclude patent licensing rights that are negotiated separately. Such licences are not recognised as Open

[39] Blind and Böhm, and Thumm (ed), *The Relationship between Open Source Software and Standard Setting*, see note 2.

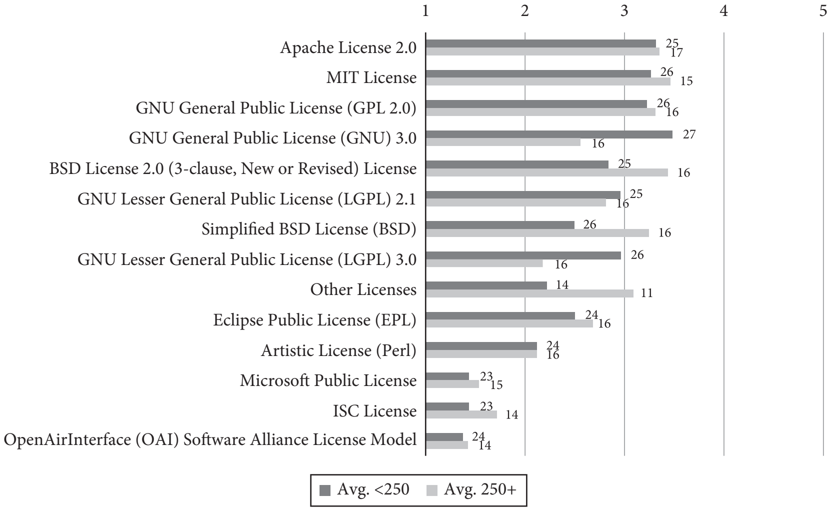


Figure 11.2 Participation in Open Source activities with various copyright licences—SMO vs LO (Scale: 1 = 'Never'; 2 = 'Rarely'; 3 = 'Sometimes'; 4 = 'Often'; 5 = 'Always')

Source: Blind and Böhm 2019, p. 159.

Blind, Böhm, and Thumm (eds), *The Relationship between Open Source Software and Standard Setting*, see note 2.

Source licences according to the Open Source Definition. Sometimes more flexible IP models are used in SDOs, which allow IP schemes and even withdrawal from Open Source software on a case-by-case basis. This is less likely than withdrawal from standardisation.

While there are no convincing constructive solutions for conflicts between Open Source and licensing models in standardisation, some approaches to general cooperation between standardisation and Open Source software are more promising, especially from the perspective of smaller organisations:

- First, the stakeholders call for greater flexibility in the patent policy of SDOs.
- Secondly, new processes for integrating Open Source into standardisation.
- Thirdly, not only is a more flexible patent policy called for, but it is even proposed that SDOs change their patent policy in the direction of licence freedom.
- Fourthly the use of Open Invention Network or equivalents with respect to the patents in the SEPs or subject to FRAND licensing.
- There are also new governance and conflict resolution models, the use of licences that explicitly exclude patent rights, and finally, a direct combination of SDOs and Open Source communities.

11.5 Conclusion

The development of the IP framework in the ICT sector is complicated by the ongoing hardware commodification, which leads to the predominant use of off-the-shelf general-purpose computers that integrate virtually all primary ICT functions such as computing, storage, networking, or telecommunications, and peripherals. Manufacturers face a market situation in which they need access to an extensive and comprehensive set of IP held by different players to produce competitive products. This means that standards and Open Source development must be analysed in combination when assessing governance standards and IP frameworks in the ICT sector. Market players are aware of this situation and have, in part, already adjusted by generally preferring consensus-oriented cooperation. Formal rules serve as a fallback for conflict resolution and may not form a practical day to day solution.

Compatibility between Open Source and FRAND licence conditions is recognised as a prerequisite for collaboration but is not communicated as a practical problem and is considered solvable. Legal compatibility is a necessity but not a sufficient condition for possible collaborations of SDOs and Open Source. Only a small number of licences are relevant in practice, which reduces the problem space for the analysis of the compatibility of Open Source software and FRAND licences.

Regarding the existence of conflicts between the various copyright licences and the licensing models in standardisation, in particular FRAND, both the GPL 2.0 and 3.0 and the LGPL 2.1 and 3.0 are mentioned by the majority of the stakeholders. Even with incompatibilities between Open Source licences and SDO IP frameworks, the stakeholders typically resolved these issues or worked around them driven by the common interest in the collaborative development of a standardised technology.

In case of conflicts, the strict separation between Open Source software and FRAND licensing is still the preferred option, followed by negotiations to find solutions supported, and if no solutions are found, in particular small organisations withdraw from standardisation.

Another less popular option is the use of copyright-only licences explicitly excluding patent licence rights, which are negotiated separately.

SDO governance focuses on the legal and IP framework and is implemented within policy constraints in a self-regulatory manner, building on the basic policies defined in interaction with policy-makers (for an overview of different SDO IP governance models see the reference by Baron and others).[40] Open Source governance is anchored in cooperation models and builds on as yet unregulated authority within the autonomous group of contributors. Open Source governance continues

[40] Baron, Contreras, Husovec, et al (eds), *Making the Rules,* see note 26.

to converge, with volunteer-led communities relying on more implicit governance standards and industry-led Open Source communities, creating more explicit rules in increasingly normalised project charters and governance structures. Governance in SDOs and Open Source communities still differ in key aspects of philosophy and implementation, which is a significant obstacle for collaboration.

SDO processes are inclusive in terms of involving a broadly defined group of stakeholders. They are also integrated into industry and policy-making. Open Source communities usually include companies, other organisations, and individual software developers without any systematic multi-stakeholder engagement. There is a strong overlap of participants in the development of standards and Open Source, especially for large companies. Overall, Open Source processes have a merit-based structure and are less accessible to policy-makers and difficult to influence in line with industrial and innovation policy objectives.

Both SDO and Open Source communities are capable of small to large collaborations (in terms of the number of participants) and small to significant R&D investments. The Open Source umbrella organisations increasingly provide platforms for cooperation and consensus building, traditionally provided by SDOs. The broader use of implementation first and parallel approaches to standardisation influences the utility of specifications concerning the value of common deployments. This changes the role of standards themselves, as standards and the development of Open Source become alternatives to achieve market dissemination for a technology. Open Source is a new challenge for innovation management as it creates an innovative, state-of-the-art technology offered with the attributes of a consumer good, with the potential for accelerated mass adoption, which have traditionally been seen as opposites and public goods.

Globalisation and online collaboration are shaping the landscape of Open Source communities and SDOs to the extent that interactions are based primarily on relevance in the respective market segment and less on formal recognition. However, formal recognition still serves a purpose as it signals, for example, relevance for security standards and for a reliable basic policy accepted by policy-makers. The converging functions of SDO and Open Source umbrella organisations offer actors a choice of platforms that did not previously exist. Both approaches are successful in providing interoperability and competitive, innovative technical solutions. In both cases, access to a wide range of technologies needed to produce competitive products is key to the freedom of action of the implementers.

Three scenarios were observed: specification-first, implementation-first, and parallel standardisation. The 'Specification First' approach is becoming less important in relative terms but is still essential in specification-driven technology areas. In particular, the parallel approach to standardisation represents some of the successful interactions between standards and Open Source development and can lead to higher quality standards, more innovation, and better implementation.

Innovation policy focuses on the framework conditions for IP and the orientation of research and development financing towards increasing competitiveness and promoting the development of technological champions and industrial competence areas. Open Source processes represent a viable additional approach for the development of technical standards. The success of Open Source communities is driven by their dynamic capacity for innovation.

12
Export Control

Mishi Choudaray and Michael Cheng

12.1 Introduction

Export control is one of many types of trade regulation that governments use to promote their economic, diplomatic, and national security interests through international commerce. Other forms of trade regulation include sanctions, tariffs, embargoes, and the closely related import control regime, among others. Similar to any other policy-making exercise, trade regulation will often have unintended side effects in areas outside of policy-makers' then-current core objectives and priorities. The distributed manner in which Open Source and other decentralised collaboration models (such as blockchain) are developed, improved, and licensed in the context of international trade will often create such side effects and continue to create novel questions of how to apply trade regulations designed for more traditional means of commerce.

The complexity of navigating trade regulation is loosely driven by trajectories of trade liberalisation and economic nationalism. Understanding how the policy-makers will wield the specific types of technologies that impact your organisation will be critical in formulating long-term plans for how your organisation will meet the challenges or capture the opportunities presented by a changing geopolitical landscape.

In this chapter, we will first present a checklist with common issues when analysing export control issues in Open Source, followed by an example of how export

Mishi Choudaray and Michael Cheng, *Export Control* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Mishi Choudaray and Michael Cheng 2022. DOI: 10.1093/oso/9780198862345.003.0012

control regimes have been applied to Open Source ecosystems in the US, and a survey of the applicable export control regimes in the UK, the EU, and China.

12.2 Export Control Checklist

12.2.1 Pre-emption and conflicts of laws

In many jurisdictions, export control regimes are pre-empted by other forms of trade regulation or other national laws and regulations, particularly comprehensive or list-based sanctions, some of which may overlap and in some cases conflict. These need to be analysed as a first step prior to starting any substantive export control analysis. Export control analysis in some cases may also need to be followed by import control analysis, as many jurisdictions outright prohibit or require licensure for the import of certain technologies or products.

12.2.2 Subject matter

Jurisdiction over what technology is subject to export control regimes is commonly based on the origin of the technology created and where additional elements were added. This may go so far as to consider where individual components originated. Governments typically assert export control jurisdiction over materials with a minimum level of technology or materials that originate domestically. The critical and sometimes challenging question becomes what exactly is the minimum level of technology that is needed to subject an otherwise foreign piece of technology to domestic export control regulations. The answer to the question typically depends on what the technology is and the export destination, among other factors. The answer may also prescribe a value threshold (the relative value of a specific component vis-a-vis an entire product) for whether or not a component is subject to domestic export control regimes.

In the context of Open Source (and with complex technology products with global supply chains), unless you are in a position conclusively and easily to identify the origin of all contributions or components, it may be most practical to assume that multiple export control regimes apply from all jurisdictions from which at least some contributions originate. Unless there is a convincing case for determining that a particular regime does not have jurisdiction, it is rarely fruitful to invest time into digging into the origin of contributors for Open Source. Moreover, the economics of Open Source development make it inherently challenging to value specific contributions.

12.2.3 Definition of export, re-export, deemed export

In general, export control regimes have taken a fairly broad view on what activities constitute an export for the purposes of falling within the jurisdiction of a particular regime. Typically, no sale or exchange is required, and even domestic disclosures of source code to foreign nationals may be deemed an export. Export control regulations also often follow the path of domestic origin technology, so transfers across international borders completely outside the domestic borders of a given regime may also be subject to a 'long arm reach' regulation. Given the broad definition of what activities would fall within export control regimes, most routine operations of Open Source projects involving the transfer or sharing of information about source code would probably qualify as an 'export' under such broad definitions.

12.2.4 Entity-based restrictions

Similar to the way that national and international sanctions work, export control regimes may also call out specific entities (including specified affiliates) that may not receive even basic or foundational technology without a specific approval or licence from the relevant government entity.

12.2.5 Exception for publicly available technology

Some export control regimes may provide specific exemptions for technology or materials that are made publicly available or produced in the pursuit of scientific research. These exemptions provide the most compelling basis on which to assert that fully public Open Source development does not fall within the jurisdiction of export control regulations. The prevailing interpretation is that in most cases, forks, pull requests, issues, merges, branches, and other activities conducted in public repositories fall within such exemptions.

It is important to note these exceptions may not apply to the private transfers of publicly developed, Open Source technology. For example, if one were to take publicly developed materials that are otherwise subject to export control regulations and then export those materials in a private or commercial context, a normal export control analysis may apply. Similarly, hybrid models of Open Source development (also known as 'Open Core', as discussed in Chapter 16) may also be subject to export control regulations to the extent that parts of their development are used for the creation of software-based commercial products.

At the time of writing, a grey area exists in the case of the development of technology that is destined to be open sourced but which either not immediately open to the public or developed or published in public (a practice still common in companies and the public sector) or where some or all of the materials are not eventually publicly made available.

12.3 Case Study: Application of Export Control Regimes to Open Source in the US

The US, where many of the Open Source projects are structurally headquartered or have large bodies of contributors, employs multiple export control regulation regimes to achieve a variety of national security, foreign policy, military, and economic purposes. Each export control regulation system is limited in scope: a schedule describes the goods, data, or persons subject to export control under the regime.

As a general matter the regulatory regimes can be distinguished between item-based and transaction-based controls. The Export Administration Regulations (EAR) and International Traffic in Arms Regulations (ITAR) are item-based while the sanctions regimes administered by the Office of Foreign Assets Control (OFAC) are typically transaction based. While EAR and ITAR are organised around the *what* of an export, OFAC is organised primarily around the *who*.

ITAR regulates the export of all defence articles (including technical data) and defence services. The United States also restricts exports to specific foreign nations and individuals for foreign policy reasons and restricts exports of some specially controlled technologies like nuclear power. These regulations are not within the scope of this chapter.

Unless your Open Source is primarily a ‘defence article’, the relevant agency is the Bureau of Industry and Security (BIS) which sits under the US Department of Commerce that administers export control regulations for software. BIS regulations: the Export Administration Regulations (EAR), arguably pose the biggest regulatory obstacle for the distribution of Open Source software. The EAR has jurisdiction over a broad scope of activities including posting of software on the Internet.

BIS is responsible for export controls on ‘dual-use’ commodities, software, and technology. An item is ‘dual-use’ when it has both commercial and military uses. There are no EAR obligations associated with the item unless it is exported (including deemed exports as described earlier), re-exported, or transferred.

These are specially defined terms in the EAR. Certain foreign-made items that contain less than a *de minimis* amount of US origin content are not subject to the EAR.

Generally speaking, a ‘US origin’ item subject to the EAR is one which was:

(1) produced or originated in the US,

(2) a foreign-made product that contains more than a specified percentage of US-controlled content (the *de minimis* rule), and/or
(3) a foreign-made product based on certain US-origin technology or software and is intended for shipment to specified destinations.

US origin items subject to the EAR, including technical data, cannot be exported or re-exported from the US without authorisation. 'Exporting' means an actual shipment, transfer, or transmission out of the US or a transfer of such software in the US to a non-US person, an embassy, or affiliate of a foreign country. This also includes downloading or causing the downloading of encryption software to locations (including electronic bulletin boards, Internet file transfer protocols, and World Wide Web sites) outside the United States. Re-exporting is the transfer of a US origin item from one foreign country to another or the transfer of a US origin item between foreign nationals.

With respect to Open Source software, foreign-made software that '*incorporates*' controlled US origin software remains subject to the EAR unless the US origin item represents a *de minimis* portion of the final product. The determination of whether particular Open Source software contains greater than a *de minimis* amount of controlled US origin software is generally based on the economic value of the US origin software and the total economic value of the finished software. Typical thresholds for *de minimis* calculations include 25 per cent, 10 per cent, and 0 per cent, depending upon the nature of the item and the intended destination.

The production of Open Source software is an international endeavour with developers based in countries around the world. Many large Open Source software projects are likely to include contributions that are considered US origin goods, potentially bringing such projects under the jurisdiction of BIS. While many Open Source software projects do make an effort to track contributors, projects do not typically record the information necessary to determine if a contribution should be considered a US origin good. Therefore, it is generally best to assume that large Open Source software projects are subject to US export controls, particularly if the Open Source software project is not publicly available. However, persons exporting Open Source software under the regulations may be able to receive authorisation to export software either by applying for and receiving a licence, or by qualifying for a licensing exception.

In contrast to software that is not publicly available, most 'publicly available' software is not subject to the EAR. Open source software distributed under an OSI-approved licence will generally be considered publicly available, because Open Source software is typically distributed in a manner that makes source and object code generally accessible to any interested party. While Open Source code itself may be publicly available and not subject to the EAR, an item is not considered publicly available merely because it incorporates or calls to publicly available Open Source code. Rather, a newly created software that incorporates other publicly available software would still need to be evaluated as a whole under the EAR.

Another caveat to the 'publicly available' exception for software subject to the EAR is publicly available software that contains or is designed to make use of encryption technology or software. This can include linking to an external application or library containing encryption functionality; software does not need to contain the encryption function itself in order to fall under the EAR. Today, the use of encryption is pervasive. Encryption functionality is standard in everything from web browsers and email clients to word processors and operating systems. Hence, the EAR has the potential to impact a major portion of Open Source software projects substantially.

Insofar as an Open Source software product contains encryption, compliance with the EAR can be achieved by notifying BIS and the ENC Encryption Request Coordinator via email of the Internet location (e.g. URL or Internet address) of the publicly available encryption source code. In addition, if you posted the source code on the Internet, you must notify BIS and the ENC Encryption Request Coordinator each time the Internet location is changed, but you are not required to notify them of updates or modifications made to the encryption source code at the previously notified location. Per instructions issued by the agency, an item is not considered publicly available merely because it incorporates or calls to publicly available Open Source code. Instead, it requires separate evaluation as a whole under the EAR because a new item with encryption functionality has been created.

12.4 Survey of Export Control Regimes

12.4.1 UK

The applicability of European export control and free movement laws to the UK, ended on 31 December 2020. The UK requires an export licence before the export of controlled military goods, software and technology and items on the UK dual-use list from the UK to another country. These items are listed in Schedule 2 and 3 to the Export Control Order 2008. Software can be assessed against the UK Strategic Export Control Lists to determine whether or not it is controlled.

Dual-use items, that can be used for civil or military purposes and associated technology and software, may be exported under national general export authorisations, such as the UK's open general export licences (OGELs—pre-published licences) that the exporter needs to register for via the SPIRE licensing database.

The Export Control Joint Unit (ECJU), part of the Department of International Trade, administers the UK's system of export controls and licensing for military and dual-use items, issues OGELs and undertakes compliance audits. UK-controlled dual-use items are specified in Schedule 3 to the Export Control Order 2008 and include related technology.

12.4.2 EU

EU export controls apply to items (including software and technology) that are dual-use items. Additional controls apply to certain specific sanctions lists which may need checked by the regime applicable to dual-use items is the export control regime most likely to be relevant to commercial software.

The categories of software that are subject to EU export control on dual-use items are listed in Annex 1 of Council Regulation (EC) 428/2009 (EU Dual-Use Regulation) (as amended, including by Council Regulation (EU) No 1232/2011) include:

- cryptography for data confidentiality having in excess of 56 bits of symmetric key length and the use of an asymmetric algorithm where the factorisation of integers is in excess of 512 bits.
- consumer, mobile phone, and end-user banking products, are excluded from export control and are listed in an Annex.

The definition of 'export' under the EU Dual-Use Regulation is broad and in addition to physical exports include transmission of software or technology by electronic media to a destination outside the EU and also includes making available in an electronic form such software and technology to legal and natural persons and partnerships outside the EU. The burden of compliance under the EU Dual-Use Regulation falls on the 'exporter' which is defined in the regulation but could include a project or individual.

The EU Commission has introduced a number of General Export Authorisations (GEA), including a GEA for the export of all dual-use items (including encryption software) to Australia, Canada, Japan, New Zealand, Norway, Switzerland (including Liechtenstein), and the US (EU001) and a further GEA to a wider range of countries for some but not all dual-use items (including some but not all encryption software protocols) (EU002).

Each of these includes:

- the destinations to which exports are permitted;
- the items that may be exported to those destination; and
- any conditions of use

GEAs are valid in all member states across the EU meaning that a company established in one EU Member State may export from that or any other Member State under the GEA providing it complies with the GEA and any additional requirements.

'Dual-use' products (such as encryption software) are not generally subject to any export controls between EU Member States.

In addition to the EU regime, Member State laws also control certain dual-use software.

The balance between the US and the EU creates some complications due to differences in approaches between the export control regime in the US and the EU. For example, US export controls include an exemption for 'mass market' items (see US 'Mass Market' guidance) that is less restrictive than the EU exemption for products that are generally available to the public (see EU Guidance note 1/2016). Software can therefore be covered by the US 'mass market' exemption but still be subject to the EU export control regime.

12.4.3 China

China updated its export control law impacting software in 2020 for the first time in over. decade. Regulations on the Administration of the Import and Export of Technology, together with the Measures for the Administration of Technology Import and Export Contract Registration and the Measures for the Administration of Export-Prohibited and Export-Restricted Technologies.

This applies to exporting controlled items produced by domestic and international branches or subsidiaries in China. It also extends to the export of controlled items by individuals and legal and non-legal entities, so having the potential reach to capture software produced by individuals, projects, and companies in China, to the extent that software is deemed to be a controlled item.

A number of emerging technologies have been included indicating an increased focus on software.

Items are split into three categories:

- Export prohibited—banned;
- Export restricted—restricted subject to the approval of the Ministry of Commerce (MOFCOM); and
- Freely transferable

The Ministry of Commerce (MOFCOM) and the Ministry of Science and Technology (MOST) maintain a 'Catalogue' of export-prohibited and export-restricted technologies. This covers transferring technology outside of China, 'in the form of trade, investment or economic and technological cooperation'.

On a *de minimus* basis the following constitute an act of export:

- assignment of patents or patent applications;
- licence of patents; transfer of know-how; and
- technical services.

Open Source software may clearly fall under this.

Like the US legislation this covers dual-use items, in other words items which can be used for military and non-military purposes. It also applies to items which can impact China's national security and interest or performance of international obligations. International treaties may well include provisions relating to software and this may well be of relevance.

The laws extend to goods, technology, services, and data that involve or relate to any of the controlled items. Export of software may require a licence, and transfers between nationals and foreign employees could be caught. Export of any restricted technology requires both a pre-approval and post-verification by MOFCOM.

A compliance program is recommended, and may be necessary for Open Source software projects incorporating contributions from China.

12.5 Recommendations

It is always advisable to hire an attorney well versed in export control compliance in your jurisdiction and train your in-house compliance teams for compliance with the myriad, dynamic regulations. Open Source software is typically contributed to and distributed in a way that makes compliance with export control record keeping requirements simple, because all source code is licensed under the terms of an OSI-approved Open Source licence and disseminated to the public without restrictions and there are few, if any, documents related to exporting opens-source software.

Nevertheless, always follow notifications and advisories from the relevant agencies and government departments for comprehensive information on export licensing requirements and policies, keep exhaustive records and ask for opinions in case of any doubt. Violations of these regulations may be subject to both criminal and administrative penalties.

13

Open Source Software and Security

Practices, Governance, History, and Perceptions

Charles-H Schulz

13.1 Open Source and Security: Myths and Reality

It is customary to read or hear that when it comes to Open Source Software, secure code is not only expected but also a reality. The problem with this kind of assertion is that any security professional will point out that appearances can be deceptive and that there is no system good enough to ensure perfect security whether Open Source or proprietary. This remark applies to pretty much anything: people, goods, buildings, monetary and financial assets, and, of course, software.

13.1.1 Software and security: an unresolved relationship

The history of computing pays little attention to the security aspects of cybersecurity, let alone software security. The emergence of computer viruses in the early 1980s was greeted with shock and disbelief, while the very notion that code should be secured in its execution, let alone written with security in mind has not prevailed until well into the 1990s.

There are several reasons for this, the most prevalent and simplest one being that the very notion that a computer could be 'attacked' by anything other than physical means was a rather far-fetched idea to many people for a long time. Other reasons range from the way computers emerged and started to be used; from the

Charles-H Schulz, *Open Source Software and Security* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Charles-H Schulz 2022. DOI: 10.1093/oso/9780198862345.003.0013

beginning computers were used in environments that were at least tightly controlled in theory: university laboratories, government, military, large companies. Even where networked computers were running, it was fundamentally a network where agents and entities knew each other and the network topology was well mapped out.

Be that as it may, the IT industry and its customers have been slow to take cybersecurity seriously; only in the late 2010s have we seen a clear spike in security spending and a strong growth in the cybersecurity ecosystem, and only in the light of massive cyberattacks, with widespread infestation of malware and political turmoil caused as a consequence.

The importance of software security, however, took an unexpected turn in the early 2000s, when the debate on the respective merits of Open Source versus proprietary software was raging. One of the arguments of the proponents of software freedom was and still is that software developed publicly and collaboratively is fundamentally more secure than software developed in a black box with no transparency or external access to its source code. The contrarian thesis was that public and open access to source code was precisely a potential segue for attackers to study the source code and exploit vulnerabilities that are baked into the software from its inception.

As it stood, however, both software development and distribution models were fundamentally at odds with proper security concerns and practices. If anything, the gap between the reality and the claimed advantages of each party was widening further down the road.

We will discuss the reality of the two assertions later in the chapter.

13.1.2 Heartbleed: the rise and fall of a model

Let's now turn this to the assertion that security management works much better in Open Source than in proprietary software. The first documented claim that one model is better than the others may be found in the Eric S Raymond's seminal book, *The Cathedral and the Bazaar*.[1,2] This was amplified later in works by renowned scholars such as Yochai Benkler,[3,4] the assertion could be summarised in the following way: given the scrutiny of 'many eyeballs', vulnerabilities and bugs can be detected early in the development and release process of software, and such scrutiny is only possible with Open Source development models because of their intrinsic public nature and openness to feedback and contribution.

It is true that both vulnerabilities and bugs may be detected early because of the open way in which Open Source gets developed and distributed, and there is

[1] Raymond, *The Cathedral & the Bazaar Musings on Linux and Open Source by an Accidental Revolutionary* (California, USA: O'Reilly Press, 1999).

[2] Raymond, *The Cathedral & the Bazaar,* see note 1.

[3] Yochai Benkler, 'Coase's Penguin, or Linux and the Nature of the Firm' (2002) 112 *Yale Law Journal*.

[4] Benkler, 'Coase's Penguin', see note 3. II. C. 'Markets, Hierarchies, and Peer Production as Information-Processing Systems'.

plenty of data suggesting that early detection of vulnerabilities and bugs happens on a regular basis for a lot of Open Source. In this case, the 'proof of the pudding is in the eating': online tools and platforms designed specifically to allow public scrutiny of the code are common practice for Open Source such as bug-reporting tools in the form of public issue trackers, quality assurance platforms designed to track and manage team work on pre-existing code, public code repositories, automated tests, unit and functional testing of software, etc. These tools are used on a daily basis by Open Source developers to improve the quality of their code, and bugs as well as vulnerabilities are identified regularly.

One could, however, object that such tools tend to be used by proprietary software, only used in a private fashion, and shared inside a software vendor's own development team. Lessons learned from the world of Open Source would however point out that the public nature and open access to the code are what brings true scrutiny, not the tools themselves.

In this way, one may rephrase the assertion in the following way: public scrutiny ensures software security and quality.

Whilst it may be true that many eyes make bugs shallow from a quality perspective, it would not take long to disprove this notion form a security perspective.

In 2014, a vulnerability was discovered in the ubiquitous OpenSSL encryption library. Dubbed 'Heartbleed'[5] this vulnerability exposed encryption keys critical to a broad variety of systems, including most servers and IT infrastructure on the planet. While the patch fixing the vulnerability was trivial, the sheer existence of the vulnerability for several months went squarely against the notion of the public nature of the code being a guarantee for software security.

In fact, the nitty-gritty details of what exactly happened seem to suggest an all too human and organisational problem. Industries kept on relying on the OpenSSL library for their most critical software and systems, yet never bothered to enquire as to who its developers were. The OpenSSL developers, in turn, were not able to attract revenues from their work and sadly, had to devote less and less time to their project in order to make ends meet with paid work on other software. OpenSSL appeared in a much cruder light: an understaffed, unpaid developer team, unrecognised and with little time to improve the codebase while the world was not watching. Quite simply, OpenSSL was a critical Open Source project with few eyeballs, well-publicised, with open access, but no-one seemed interested in improving its security and quality. The initial assertion that Open Source does security better showed serious cracks.

In the aftermath of the Heartbleed affair, the Linux Foundation joined efforts with industry and US government to create the 'Common Infrastructure Initiative' designed to ensure that people working permanently on critical low-level Open Source would receive resources and constant security oversight to ensure that the code is adequately maintained on a go-forward basis.

[5] Reference CVE description on the Mitre's CVE base: CVE-2014-0160.

The Heartbleed vulnerability was in many ways a critical moment for many in the software industry to recognise that Open Source is not inherently safer because of a software licence or the mere fact of open access to the source code. Rather, it became clear that vulnerabilities may affect Open Source just like proprietary software and that the openness of the code speaks in favour of the trust we can have in it, not in inherently better security.

As we will see in the next section, Open Source projects have been practising security management and devised processes to deal with vulnerability, disclosure and patching for many, many years. The results of such a practice have led to a much-improved way to deal with security issues and better overall software quality. In this way, it is possible to revisit the assertion that 'given many eyeballs' software can be made more secure.

Unfortunately Heartbleed was not the end. In December 2021, the Apache Log4J vulnerability "Shell4J" had global impact. Proprietary software platforms discovered it in their codebases as well as it impacting Open Source users. A perfect case study in supply chain transparency. Shell4J, the Biden Ordnance, war in Ukraine/bad actors and geo-political shift all impact governments globally. National infrastructure, including critical national infrastructure has digitalised. It is software defined, therefore built on Open Source. Governments are scrambling to catch-up, taking steps to secure all software infrastructure including Open Source.

In May 2021, the Biden Administration issued an Ordnance (Biden Ordnance)[6] on Open Source software security, supply chain and use of Software Bill of Materials (SBoMs), discussed in Chapter 8, creating a dramatic increase in use of SBOMs in the US. In 2022, a draft Open Source regulation has been tabled in the US[7] and in Europe the Cyber Security Act[8] also lifts much of the Biden Ordnance. The Open Source Software Foundation (OpenSSF)[9] held meetings with the WhiteHouse through 2021/2[10] with 20 top vendors. Its outputs—a 10 point action plan, multimillion dollar funding for the "Alpha Omega"[11] project from Google and Micorsoft supporting improved security for example funding the Rust Foundation and international activities like OpenUK's Summer of Open Source Software Security.[12]

Users of Open Source are responsible for its curation—ensuring it is well implemented, governed and maintained/kept secure. Open Source software producers can also support this in their practices. Curation and security will continue to be a key focus beyond the date of writing of this chapter.

[6] https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

[7] https://www.govinfo.gov/content/pkg/BILLS-117s4913is/pdf/BILLS-117s4913is.pdf

[8] https://digital-strategy.ec.europa.eu/en/policies/cybersecurity-act

[9] https://openssf.org/

[10] https://www.whitehouse.gov/briefing-room/statements-releases/2022/01/13/readout-of-white-house-meeting-on-software-security/

[11] https://openssf.org/community/alpha-omega/

[12] https://openuk.uk/security/

13.2 Open Source Security Governance: Vulnerability Discovery, Patching, and Disclosure Practices

There is something of a paradox in discussing security processes and vulnerability disclosure for an Open Source project. After all, one might be excused in thinking there would be very little of that in an environment where everything is supposed to be open and public. In practice however, openness is always about the code, but not necessarily about the process.

It is important to note that the following description below applies mostly to well organised and somewhat large Open Source projects. It is not necessarily the case for smaller scale projects with less resources, fewer developers, who are less used overall.

13.2.1 How is security managed in Open Source projects?

In fact, there are and have been security management processes implemented inside Open Source projects at least since the late-1990s.[13]

What these security management processes entail are made of roughly four elements:

- a dedicated, invite-only security mailing list;
- a commitment to handle and fix vulnerabilities either urgently or assign them a high priority (unless they are not considered non-critical);
- some coordination with the broader ecosystem in order to identify and understand better how some common form of vulnerabilities or even ongoing attacks affect not just the project but a broader range of software; and
- some form of active security watch or bug-hunting program that can be conducted on a regular or even continuous basis.

Each of these four elements must be reviewed in a bit more detail.

13.2.1.1 A dedicated, invite-only security mailing list

This is perhaps the simplest and oldest tool to manage the security of Open Source projects. Such a mailing list may be run in several different ways, from a community's core developers discussing and interacting with people who may have found a vulnerability to a closed list receiving input from a public, dedicated email alias which anyone can write to in order to send a vulnerability report. This is the first level of coordination for security management and such a mailing list is essentially the only or the main place where security vulnerabilities are discussed and actual remediation is considered.

[13] Reference CVE description on the Mitre's CVE base: CVE-1999-0077.

13.2.1.2 A commitment to handle and fix vulnerabilities

This commitment is a bit trickier to explain as it relies on a process involving mostly the expertise of the project's own developers, their own resources, and their actual assessment of each vulnerability. While as a general rule there is a strong commitment from Open Source projects to fix security vulnerabilities found in their software, the criticality of the vulnerability and the impact analysis of the project's developers really come into play. Also this particular process that highlights the practical rift between the assertion that given enough public scrutiny any Open Source code can be kept secure enough. In reality, things tend to play out in a much more subtle way.

The commitment to fix critical flaws or vulnerabilities should in any case be considered as a given for every project. Whether the developer(s) can fix it or believe the vulnerability is critical enough to fix is, however, a different matter altogether.

To start with, the developer may not have enough time or even skills to fix the part of the code affected by the vulnerability. This was essentially the problem with the Heartbleed vulnerability. It wasn't a matter of skill but time, as the vulnerability was fairly trivial. The development team, however, had overlooked it for months and had not conducted any particular code analysis that would have allowed them to detect it. In this case, a combination of lack of time, resources, and interest.

The second level is the developer's understanding of the vulnerability. Once the vulnerability has been reported, the developer has to assess the criticality of the vulnerability. If the vulnerability is deemed to not be critical or not critical enough, the development of a patch and its release may be delayed, and sometimes in a substantial way. The developer's own understanding may obviously be seen as a point of failure in an Open Source project's security process. It is nonetheless the direct consequence of the developers' own independence.

13.2.1.3 Coordination with the broader ecosystem

While this point is by no means mandatory, several Open Source projects tend to coordinate with others on security matters. It is particularly true when projects have some form of interdependence or share some similarity in purpose or code. Examples may be found among Linux distributions, Open Source licensed web browsers or office suites. This coordination is quite useful and allows projects to share newly discovered vulnerabilities and then to coordinate the release of patches.

13.2.1.4 Active vulnerability watch

Some larger-scale Open Source projects have started to implement active vulnerability watch processes such as regular bug-bounty campaigns. These processes tend to be effective in identifying existing vulnerabilities, but they demand resources and the availability of the project's developers to deliver security patches on a near-constant basis. While bug-bounty campaigns and other similar active security watch processes are a growing trend, they are by no means considered a standard security process among Open Source projects.

13.2.2 Confidentiality and disclosure

It may seem surprising to discuss confidentiality of security-related information in the context of Open Source communities. There is however a good case to be made for keeping some part of security related information confidential, at least for some period of time.

One should start by defining why specific information related to vulnerabilities is not immediately disclosed and kept private. Software security circles and people are not necessarily the same as the ones found in Open Source projects. As such, the culture may not be exactly the same and publicity of code may not be seen as a value in itself. More precisely, there is an understanding that timing is an important element in software security. Regardless of its licence, any given software may have published vulnerabilities (patched or left unpatched) that are documented; it may also have undocumented and unreported vulnerabilities—also known as 0 Days—that have been discovered by one or several attackers. It may also be discovered by a developer or security researcher and as a result it could and should be reported and documented. The exact interval between a vulnerability report and its disclosure—usually quickly followed by a proper fix—is a time when information about the vulnerability is considered private or even confidential.

This allows for the project's developers to run their own assessment of the vulnerability properly and prepare a proper patch. It also helps in obfuscating the very existence of the vulnerability to the public, thereby reducing the risk its exploitation by even more attackers.

Over the course of time, vulnerability disclosure models or processes have been established, leading to a process usually known as a 'Responsible Disclosure' process. While Responsible Disclosure is a mainstream practice, there are security experts who oppose it and favour a model in which the vulnerability is disclosed at soon as it is known by the Open Source project. This process is known as Full Disclosure but is seldom used.

Responsible Disclosure may certainly be understood as a departure from the full public model of Open Source projects where code development, bug-tracking and releases are essentially fully open and public. It is, however, an exception to the openness rule that is limited both in time and scope and allows Open Source projects to interact with the broader IT ecosystem. Software security remains a somewhat particular field even for Open Source and a possible way to understand its constraints is that just like in the physical world, security is primarily a collective matter. As such, proper cybersecurity can adapt quite well to software freedom and Open Source, but still has its own imperatives.

PART 2

THE BUSINESS OF OPEN: ECONOMICS, OPEN SOURCE MODELS, AND USAGE

14
Sustainability and Open Source

Cristian Parrino

Sustainability is the challenge of our time and the problem set is broad: increased inequality; deepening social and financial exclusion; climate change; air, land, and ocean pollution; the ecosystem breakdown that can lead to pandemics. These problems are all related and share the same root cause: an economic model which is relentless in its pursuit of perpetual growth at the expense of people and planet.

In 2015, the United Nations General Assembly published the Sustainable Development Goals (SDGs) as its blueprint of seventeen interlinked global goals to tackle this generational challenge and provide a better and more sustainable future for all, intended to be achieved by 2030.[1]

The transformation required isn't just about decarbonisation but is a systemic transformation, one which starts with shifting to an economic model that prioritises prosperity, which economist Kate Raworth defines in her book *Doughnut Economics* as the aim of meeting the needs of all people within the means of the living planet.[2] Doing so involves creating a regenerative and distributive economic model which ensures that a social foundation is met for people across health, education, income and work, peace and justice, political voice, social equity, gender equality, housing, networks, energy, water, and food, while at the same time staying within an ecological ceiling that protects our planet from climate

[1] <https://www.un.org/sustainabledevelopment/sustainable-development-goals/> accessed 16 April 2022.

[2] Kate Raworth (ed), *Doughnut Economics: Seven Ways to Think Like a 21st-Century Economist* (London: Random House, 2017).

Cristian Parrino, *Sustainability and Open Source* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Cristian Parrino 2022. DOI: 10.1093/oso/9780198862345.003.0014

change, ocean acidification, chemical pollution, freshwater withdrawals, land conversion, biodiversity loss, air pollution, and the depletion of the ozone layer.

Given the multi-decade complexity required for this kind of transformation, it is imperative that the Open Technology principles of collaboration, decentralisation, and collective equity, as well as Open Technology's (hardware, software, and data) fundamentals of transparency, circularity, and accessibility play a central, enabling role.

14.1 From Human-Centred Design to Community-Centred Design

Many of the valued and accepted innovation frameworks used to develop services, products, and standards such as human-centred design and design thinking view people solely as individuals or, even worse, consumers. They are focused on identifying individual user needs and how they can be fulfilled with more products and stickier services, and the role of technology is primarily to drive operational efficiency, hockey-stick growth and profit. These innovation frameworks are designed from a position of privilege, conceived by Ivy League academia, made popular by Silicon Valley and their venture capitalists, and turned into toolkits by big consultancies, all of whom are more often than not a white, male, and from an affluent background. The result is that people aren't considered in relation to their communities, and society itself isn't considered at all. At a time when we must design solutions that take into account the people and planet systems that we impact, individual-centric innovation frameworks that design out society cannot be accepted.

Creating products, services, and standards that are fairer, more inclusive, and empowering for people, which design out structural inequality and are regenerative towards the planet requires a move towards a community-centred design model, underpinned by cooperation, transparency, decentralisation, and collective equity; in other words, an Open Technology model.

14.2 The City of Amsterdam Case Study

The City of Amsterdam is using Open Technology and community-centred design to rebuild its economic model and post-pandemic recovery around the principles of economic prosperity. In April 2020 it became the first municipality in the world to publish—as an Open Technology framework—its 'City Doughnut', and its plan to emerge from the COVID-19 crisis as a city that ensures a good life for everyone, within the earth's natural boundaries. The vision is to transition Amsterdam into a *circular city*—one in which waste and pollution are designed out, products and materials are kept in use for as long as possible, and natural resources are regenerated—adopting open and smarter approaches to managing scarce raw materials, production, and consumption, and creating more green jobs.

The Amsterdam City Doughnut is the open tool that is being used to drive this transformation.[3] It provides a holistic view to inform policy decisions and implement circularity which will initially focus on its three priority value chains: food, consumer goods, and the built environment.

These were identified by the city as the three most impactful value chains in terms of economic and ecological impact. A third of the city's food is wasted, the use of consumer goods is responsible for a massive ecological footprint, and 40 per cent of all waste produced in the city is generated by the built environment.

Across each of these value chains, the city outlines interventions around shortening and localising the supply chain, reducing natural resource use and consumption, and making the most of waste streams. Over 200 organisations in the city are openly collaborating on Amsterdam's transformation.

14.3 The Emissions Problem and the Role and Complexity of Supply Chains

The World Economic Forum estimates that eight supply chains are responsible for more than 50 per cent of global greenhouse gas (GHG) emissions.[4] These include food, construction, fashion, fast-moving consumer goods (FMCG), electronics, automotive, professional services, and freight. Emissions generated by the supply chain are known as Scope 3, which include upstream emissions from procured products, transport of suppliers, and business travel, as well as downstream emission from the transport of products, usage of sold products, and product disposal. In Scope 1 are the emissions generated by the operations under an organisation's control, while Scope 2 comprises the emissions generated from the use of energy purchased from third parties. Across these eight supply chains, Scope 3 represents 77–95 per cent of an organisation's true cost in terms of GHG emissions. Businesses have largely focused on reducing Scope 1 and Scope 2 emissions, in part because of the complexity of supply chains, and in part because of a misperception of the costs associated with transforming them into zero-emissions supply chains.

According to the same World Economic Forum report, net-zero supply chains would hardly increase end-consumer costs. Around 40 per cent of all emissions in these supply chains could be abated with readily available and affordable levers

[3] <https://www.kateraworth.com/wp/wp-content/uploads/2020/04/20200406-AMS-portrait-EN-Single-page-web-420x210mm.pdf> accessed 16 April 2022.

[4] <http://www3.weforum.org/docs/WEF_Net_Zero_Challenge_The_Supply_Chain_Opportunity_2021.pdf> accessed 16 April 2022.

such as circularity, efficiency, and renewable power, with only marginal impact on product costs. Even with zero supply chain emissions, end-consumer costs would go up by 1 to 4 per cent at the most in the medium term.

Decarbonising supply chains is hard as organisations struggle to get the data needed to set the targets and standards that their suppliers must follow. Supply chains are fragmented across geographies and legislations, so decarbonisation requires open collaboration and action at the industry level.

Open principles and technology must be the basis of the transparent framework for the target setting, emissions disclosure, and progress dashboards for companies to decarbonise their supply chains. They are also essential for the enablement of circularity in order for products and materials to be kept in use as long as possible in order to decrease reliance on new natural resources.

14.4 The Carbon-Negative Data Centre Case Blue Print

The role of open hardware, software, and data in the carbon-negative data centre of the future blueprint being proposed by an OpenUK-led international consortium at COP26 is a comprehensive demonstration of how Open Technology is fundamental to enabling the circularity required to achieve dematerialisation and decarbonisation across the backbone of the information and communication technology (ICT) industry.

The Shift Project estimates[5] that on its current path, the global ICT industry could be on track to grow from around 4 per cent of global GHG today to nearly 8 per cent by 2025. The gradual shift to renewable energy, while important, is a small part of the solution. There are more than 50 million tonnes of e-waste produced annually—this number is growing and we are running out of the earth's minerals required for electronic components.

Data centres are the backbone of the ICT industry and their scale is important and immense. The International Energy Agency (IEA) estimates[6] that demand for data and digital services is expected to continue its exponential growth over the coming years, with global Internet traffic expected to double by 2022 to 4.2 zettabytes per year (4.2 trillion gigabytes). At the same time, the number of mobile Internet users is projected to increase from 3.8 billion in 2019 to 5 billion by 2025, while the number of Internet of Things (IoT) connections is expected to double from 12 billion to 25 billion.

[5] <https://theshiftproject.org/wp-content/uploads/2019/03/Lean-ICT-Report_The-Shift-Project_2019.pdf> accessed 16 April 2022.

[6] <https://www.iea.org/reports/data-centres-and-data-transmission-networks> accessed 16 April 2022.

It is clear that the ICT industry is going in the wrong direction on carbon and climate mitigation. Bold and responsible action is urgently needed and the opportunity to do so clearly emerges: Operational phase energy (Scope 1 and 2) efficiency is essential, but only a small piece of the equation. Scope 3 emissions in the supply chain also require intense focus and mitigation and, in this case, via circular data centre infrastructure and corresponding solutions.

The answer is in wholly circular, edge-based carbon-negative data centres based on the Three Opens of technology (hardware, software, and data)). The data centre needs to be fully transparent, particularly as many of the organisations that are announcing plans for their own net-zero data centres are doing so in silos and in a closed manner so processes, insights, and economies of scale are seldomly shared and implemented with all industry constituents in the interest of the collective solution.

The OpenUK blueprint is based on the circular principles of designing out waste and pollution, keeping products and materials in use, and regenerating natural systems in the process, focusing on three key pillars:

1. Decarbonisation of all key layers of the data centre. For Scope 1 and 2 direct emissions, this includes reuse of existing and abandoned buildings, full energy efficiency in the built environment, powered by renewable energy sources, local or district level heat-recovery solutions, and the electrification of the transportation fleet and of onsite fossil fuel processes such as steam. For Scope 3 indirect supply chain emissions that require a circular solution, the blueprint calls for the reuse of servers and components to the fullest extent, onsite and offsite repair and lifecycle extension of server and network hardware, reduction and offset of product transportation miles, and responsible packaging solutions. Then, sharing the learnings linked to reuse, disassembly, reassembly, and recycling to all partners in the supply chain by using creative commons for the blueprint and outputs themselves.
2. Adoption of an open technology platform across hardware, software, and data (the Three Opens) and carbon accounting frameworks used as the critical backbone in enabling circular, carbon-negative data centres. Open Source software enables release of licensed and copyrighted source code; open hardware enables the release of the designs of tangible artifacts, and open data relies on the notion of transparency and that data should be freely available to everyone. All rely on the principles of collaboration and public benefit which is key to an effective circular economy.
3. Commitment to increased transparency, inclusive economy, and equitable access—all are key components needed to build trust within communities and transform digital infrastructures into a holistic solution that contribute to sustainable cities (SDG 11). For example, the waste heat created from powering rows and rows of servers in a data centre can be recycled into homes, reducing the data centre's use of electricity for cooling, while at the

> same time decreasing the reliance on natural gas to heat homes and lowering heating costs for marginalised communities who could benefit from the local proximity of an edge-based data centre infrastructure which donates its heat outputs back to the grid in the local district.

14.5 The UN Sustainable Development Goals and Open Data

The seventeen SDGs were originally developed by an open working group of thirty member states, facilitated by the United Nations, who worked together to develop a set of action-oriented goals designed to balance the economic, social, and environmental dimensions of sustainable development. An open application programing interface (API) and open database,[7] maintained by the United Nations Statistics Division, underpin the SDGs to help member states achieve the goals and measure progress in meeting them. It provides critical information on natural resources, government operations, public services, and population demographics. For the SDGs, open data is a facilitator of standards, a tool for accountability, and a base of evidence for assessing impact.

According to a World Bank report on Open Data for Sustainable Development,[8] open data plays a critical role in fostering economic growth and job creation, improving the efficiency and effectiveness of public services, increasing transparency, accountability and citizen participation, and facilitating better information sharing within government to improve resilience and deploy resources effectively in emergency situations. The report outlines the role open data can have in establishing standards, impact assessments, and accountability:

> *Standards*: By encouraging open data standards, sustainable development initiatives can build from existing datasets, schemas, and databases to contribute to the broader evidence base. There is a need to establish a global consensus on principles and standards to be able to compare data across sectors. An example is Icebreaker One,[9] which gathers data from financial markets, public sector institutions, asset owners, and the scientific community to enable global data marketplaces that will help investors deliver innovative financing to address the climate and biodiversity crises. Similarly, the Linux Foundation announced its Climate Finance Foundation (LFCF) which is building the OS-Climate platform,[10] an initiative with the goal of empowering investors, banks, insurers, companies, governments, non-governmental organisations (NGOs), and academia with

[7] <https://unstats-undesa.opendata.arcgis.com accessed> 16 April 2022.

[8] <https://pubdocs.worldbank.org/en/741081441230716917/Open Data-for-Sustainable-development-PN-FINAL-ONLINE-September1.pdf> accessed 16 April 2022.

[9] <https://icebreakerone.org/mission/> accessed 16 April 2022.

[10] <https://www.os-climate.org> accessed 16 April 2022.</FN>

artificial intelligence-enhanced open analytics and open data to address climate risk and opportunity.

Impact assessment: Facilitated by standards, open data can help gauge the impact of development initiatives over time, geographies, and topical areas. For example, open data can help establish benchmarks to measure progress against the SDGs, both within each country and between countries. It can reveal inequalities and disparities in income, wealth, and access to government services and provide a basis for assessing progress over time.

Accountability: By releasing open data about a full range of SDG initiatives, government institutions can show their commitment to the SDGs and hold themselves accountable for the results. This transparency and accountability can also help engage citizens in working on the SDGs.

For example, for SDG-14 (Life Below Water), government data can be combined with citizen science in order to monitor fishing, pollution, and water quality. For SDG-5 (Gender Equality), open data can expose shortcomings in the way that health, education, and business serve women and girls. For SDG-7 (Affordable and Clean Energy) open data can be used to monitor consumption, energy sources, and distribution hops in order to help shift to locally produced affordable, clean and renewable sources, as well as identify ways to extend or decentralise the grid to ensure inclusive access. For SDG-11 (Sustainable Cities and Communities), open data is an essential tool for rethinking and improving urban infrastructure, as we are seeing across many smart city initiatives around the world. For SDG-12 (Responsible Consumption and Production), open data can track geolocated consumption and production patterns in order to identify ways to redistribute excess to solve shortages elsewhere and regulate overproduction and waste creation. For SDG-8 (Decent Work and Economic Growth), which would be better served by a Decent Work and Economic *Prosperity* headline, open data can be the tool that helps develop green job skills and support entrepreneurship by identifying the inclusive and regenerative products and services required to shift wealth back into local communities.

The UN's SDGs set an ambitious agenda for progress on the world's most challenging problems by 2030. The problem set is broad, complex, and requires urgent action, and open data has the potential to be a universal resource to help achieve and measure the SDGs. Combined with open collaboration, open data can be fundamental in accelerating progress against the SDGs.

15
Economics of Open Source

Mirko Böhm

The relationship between Open Source and economics is fundamental since the collaborative creation of software and its utilisation are economic activities. There is a value that businesses can generate with products that include or consist solely of Open Source, and a potential cost saving in its use. The work of the wider Open Source community is coordinated and software is created by different elements of the Open Source ecosystem. It is integrated as an intermediate or final product into consumer applications that deliver concrete, useful functionality. Open Source is unique in that it is simultaneously state-of-the-art technology, a commodity, and a public good. Open Source communities are social groups of individual and organisational contributors that participate voluntarily in the production of public information goods.

Communities are able to resolve issues without the coordination provided by a central authority like the state. Overall, the wider Open Source community

Mirko Böhm, *Economics of Open Source* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Mirko Böhm 2022. DOI: 10.1093/oso/9780198862345.003.0015

contributes positively to the common good. This shapes how Open Source collaboration relates to market competition and the value propositions of businesses. Today, collaboration spans both unpaid or volunteer participation and industry contributions that are made in the course of employment or under the guidance of a commercial sponsor. This chapter develops a basic taxonomy based on a combination of the revenue model, the type of good, and the differentiating aspects of Open Source-based products. It positions Open Source in its relation to economics and discusses the different behavioural norms like reciprocity and fairness that participants apply to the social transactions of Open Source collaboration, as well as the impact of Open Source on the technology stock of society.

15.1 The Economics of Open Source

In recent years, Open Source has gone through a remarkable transition from an exotic pastime of idealists into a mainstay of software engineering practice. Today's communities are a diverse mix of individual volunteers and industry contributors who collaborate on software development, while simultaneously competing intensively in business. Open Source is used pervasively throughout the information and communications technology (ICT) sector. Such pervasive adoption is inevitably driven by sound economic arguments that appeal to many actors with differing motivations and self-interest, indicating that there is a theoretical foundation for the mass appeal of Open Source collaboration and for the reconciliation of self-interest or business rationale with collaboration on the development of Open Source technologies.

15.2 Introduction: Open Source, Law, Politics, and Economics

Open Source itself is first and foremost source code and therefore software. This is the way many people see it—as an amazing pool of free software modules to select and build upon. However, that is not the whole story. Free licensing of source code combined with collaborative and accessible development processes create a relationship between Open Source and a cross-section of society. Richard Stallman, the founder of the Free Software Foundation (FSF), insisted that 'free software is a social movement'.[1] Software is a form of technology with wide-ranging social implications. In a society which has digitised, it impacts a diverse range of issues, including civil liberties and human rights, access to the means of production of goods and services, methods of collaboration, and many others. Three of these

[1] Richard M Stallman and Lawrence Lessig, *Free Software, Free Society: Selected Essays of Richard M. Stallman* (SoHo Books, 2010).

relationships that have attracted particular attention in recent years are those between Open Source and law; politics, and economics.

The interaction between Open Source and law manifests itself in Open Source licensing and compliance. Source code that is created by humans is covered by copyright, as explained in Chapter 3. In most cases, explicit permission from the copyright owners is required to use the code due to the application of copyright. Open Source licences are the mechanism by which authors give permission to third parties to use their work. These licences form the basis of the legal relationship between the authors and the users. Disagreements between the owner/licensor and licensees are resolved by recourse to legal processes. Other legal frameworks that intersect with Open Source include the potential to own and infringe patents in the software in some jurisdictions, as discussed in Chapter 10. Chapter 9 explores the use of trademarks to designate product origin. Contributors sometimes appoint fiduciaries that represent their rights or enter into agreements with communities or businesses and these are explored in foundations at Chapter 18. Questions also arise around liability for code which is either negligently or maliciously constructed. Issues around rights and obligations around the development, distribution and use of Open Source today are dealt with primarily through private law.

How Open Source and politics relate is possibly more abstract, but already hinted at in Stallman's statement quoted earlier.

Open Source initially challenged how the software industry innovated, by shifting from a proprietary or closed to an open and collaborative model. As software has become increasingly important in many different sectors of the economy, through a process of digitalisation, the effects of Open Source innovation have become pervasive. In particular, the collaboration methods developed by the wider Open Source community have inspired related changes in business activity around software development and also inspired changes in areas not directly associated, such as open access in science, open data, or open hardware (see Chapter 23). As part of this shift in how knowledge is transferred and monetised and how technical standards are developed and adopted, Open Source has offered new and alternative approaches to innovation.

Other changes go deeper and reflect technological development over the last two decades. Ubiquitous Internet access combined with Open Source enables previously unfeasible participatory forms of decision-making in society, alternative approaches to knowledge transfer, and opportunities for provisioning public information and communications technology (ICT) infrastructure, with reduced lock-in to specific technology providers. The gig economy has transformed the labour market and impacted employment patterns and worker mobility. Understanding of the societal changes caused by the Internet with resilient connectivity, convergence, and software freedom is still at an early stage, which is why this chapter focuses primarily on the *micro*-economic effects of Open Source.

The relationship between Open Source and economics is fundamental. Economics studies how our societies produce and trade goods and how individual actors make decisions when participating in that process. From an economic and, probably, a legal perspective, software is considered to be a good. When considering economics, it would traditionally need to be traded to be useful.

Development of software and in particular Open Source is an inherent part of the economy. In the case of Open Source its contributions to gross domestic product (GDP) may be difficult to measure since using it is generally not accompanied by a monetary exchange (although there is no prohibition on such it is not the norm).

Open Source is a software good with specific properties. In particular, it is available without significant restrictions and in unlimited quantities and allows every interested party to use and improve the existing source code, build upon it, and redistribute that modified version. This raises two key questions:

- What incentivises an individual to consume and to contribute to Open Source?; and
- What creates the balance between supply and demand so that the market is provided with the software that is needed in the necessary quality and quantities?

From a macro-economic perspective, a choice that affects us all is how society may react to the changes that Open Source imposes on it. They could be *rejected*, for example because they threaten jobs in established businesses. They could be *tolerated*, because the benefits of innovation outweigh the potential costs. Or Open Source could be *facilitated*, invested in, protected, and supported because of a belief that it is a beneficial pillar of the digital society. To make that choice, it is necessary to understand the overall impact of Open Source on society.

15.3 Why is Free Software Free?

There is an unfortunate confusion based on the meaning of the term *free* in the English language. It means both free as in free of charge as well as free as in freedom or liberty. In the case of Open Source, it is free as in liberty, but generally also free of charge, licence fee, or royalty. To understand the economics of Open Source, we must also consider not only what makes Open Source free but also why our society embraces a model where important software technology is developed in an open, collaborative model.

Software begins its life as the human-readable source code. Without the application of intellectual property (IP) protection in the form of copyright, source code can be regarded as merely information. Information is produced, traded, and has value, making it a *good* in the economic sense. Information however has properties that make it special, in particular *intangible*. It is 'costly to produce but cheap to reproduce':[2] today, reproducing (making a copy of) information on a computer practically incurs no cost at all and, increasingly, all information relevant to human activity and existence is represented in digital form on computers.

When humans consume information, for example by reading source code or experiencing a piece of music, they convert it into knowledge, a process that cannot easily be reversed (some say information is difficult to dispossess). Information generally available on the Internet can be consumed by any interested user, making it *non-excludable*. Each user's experience will be mostly unaffected by the fact that others are consuming the same information at the same time, making it *non-rivalrous*. Products that both non-excludable and non-rivalrous are defined as public goods. Information without property protection is either secret or potentially available to everybody.

There is a dilemma in that it takes effort and creativity to produce valuable information while it is easy and cheap to reproduce it. This is well-understood, and one of the foundations of the Berne Convention,[3] a pillar of international copyright law ratified by all developed countries in the world. It posits that authors acquire copyright on their works as soon as they are 'fixed', and that others need explicit permission (a licence) from the copyright owner to use, reproduce, and distribute their works. By giving authors a legal instrument to manage who has access to their works, copyright provides the framework that makes intangible information goods tradeable.

In most traditional uses, copyright and its licensing are applied to restrict the number of available copies of a work. In a competitive environment, a good that is available with unlimited supply will converge on a price of zero. This fact incentivises rights holders to limit the number of copies available in the market. For example, reproductions of paintings may be limited and books printed in batches. Binding the information good to a medium for transport illustrates its intangible nature. The number of copies of the medium is restricted as a means to limit supply and to maintain a non-zero price. If the cost of creating another copy is very low, as for example with digital music streamed from the Internet, subscription models are an example of a tool to generate revenue based on the aggregated market demand.

Open Source moves a step forward in the application of copyright to digital goods. It applies the same concepts of authorship and licensing discussed so far and applies terms that make the software available in *unlimited supply*. All Open Source licences guarantee that users have the rights to use, study, modify, and redistribute

[2] Carl Shapiro and Hal R Varian, *Information Rules: A Strategic Guide to the Network Economy* (Cambridge, MA: Harvard Business Review Press, 1998).

[3] <https://www.wipo.int/treaties/en/ip/berne/> accessed 11 March 2021.

the software. Since everybody and anybody can redistribute the code, any piece of Open Source is available to the general public once released and very difficult (if not impossible) to retract. That means a piece of software released under an Open Source licence begins life as a public good, and is made into a private good by the copyright acquired by the author, and then reverted to a public good again by the application of the Open Source licence.

However, since users have the right to use the code on the terms chosen by the author, these may contain obligations or restrictions. For example, attribution, as in naming the authors of the used Open Source building blocks in all reuse is a minimum requirement in these licences. The class of reciprocal, 'copyleft', Open Source licences require that users distribute their own modifications under the same terms, ensuring that the software remains Open Source even if modified.

The use of traditional copyright licensing in a way that enables sharing and user freedoms, effectively playing the IP right at its own game to revert its impact, is a 'stroke of genius' attributed to Richard Stallman. As a consequence of this action, all Open Source licences are anchored in the author's copyright and precluded the need for an Open Source-specific legal framework or 'lex Open Source'.

15.4 Software Freedom and Open Collaboration

The basic concept of Open Source mixes two perspectives, that of free software as a product and that of open and collaborative processes for the development of software. A typical understanding is that an Open Source product is both free and also developed in an open, transparent collaborative process. These two perspectives are distinct.

Software is considered Open Source if it is made available under an accepted Open Source licence. Today's understanding of the formal requirements for code to be Open Source builds upon the 'four freedoms of software' as laid out by the Free Software Foundation (FSF)[4] and more generally the Open Source Definition (OSD) maintained by the Open Source Initiative (OSI).[5] All Open Source licences give the users of the software at least four essential rights, namely to use, study, modify, and redistribute the software without discrimination between who uses the software and for what purpose. This definition does not prescribe in what way the software is created.

Open Source contributors may not consider their works to be products since they are not sold in the market. In the context of this article, the term 'product' is understood as a good made available for another's use. The requirement is that having been produced the good is made available, not that it is being sold at a price.

[4] <https://www.gnu.org/philosophy/free-sw.en.html> accessed 11 March 2021.

[5] <https://opensource.org/osd> accessed 11 March 2021.

Various goods are made available for free even though they are costly to produce, for example in state-provided free education.

15.4.1 Methodologies

Open Source may be created in a community process where interested parties may participate and contribute to its production based on the merit of their contributions. What if any requirements of this *openness* exist is still the subject of an ongoing debate. In most communities, it means that all contributors are welcome and should be treated respectfully and equally.

However, there are businesses that maintain control over an Open Source product or a commercial version of it, while accepting outside contributions. A company may act as a commercial sponsor to an Open Source product, usually combining it with offering a complementary commercial version or support, such as Canonical sponsoring the Ubuntu operating system.

Another form of development is the release of projects originally developed in-house under an Open Source licence, like Google's Kubernetes.

Many projects today start as industry collaborations where businesses cooperate on the development of a foundational technology or an industry standard. Such projects are often set up at Open Source foundations, as elaborated in Chapter 18.

Some refer to this by saying 'there is more than one FOSS [Free and Open Source Software] way'.[6] Since this Open Source governance is less standardised than Open Source licensing, it is assessed indirectly by measuring accountability, transparency, or the accessibility of the community decision-making processes.[7] Calling software Open Source conveys certain positive values on the code and may be used by businesses in marketing, whether accurately or otherwise, adding to the confusion.

The distinct merits of Open Source products and open collaboration are each economically relevant. Similar to inventions, Open Source *products* become part of the technology stock of society and influence the state of the art of products and production processes. Due to their free nature they may be the subject of both rapid adoption and ubiquity.

Even though Open Source is usually distributed freely, its functionality has an impact that has value and therefore impacts GDP. Open Source development *processes* enable efficiency gains that contribute to economic growth for example through improved interoperability or as a consequence of their free adoption may eliminate duplicated efforts.

[6] <https://opensource.com/Open Source-way> accessed 11 March 2021.

[7] Mirko Böhm, 'The Emergence of Governance Norms in Volunteer Driven Open Source Communities' (2019) 11(1) *Journal of Open Law, Technology & Society* 3–39.

15.5 Differentiate or Collaborate!

According to the Maddison project database,[8] real GDP per capita in Germany increased more than twenty-fourfold between 1850 and the year 2000. Other industrialised countries show similar increases.

GDP per capita depends primarily on the technology stock applied to production since it does not increase if more people produce the same amount per person. It is plain to see that the dramatic increases in the standard of living experienced in this period depend heavily on technical innovation. While it is difficult to provide a clear definition of what innovation is,[9] its effects are real. It is the improvement of the technology stock available in an economy that leads to increases in real GDP per capita and lays the foundation to improve the general standard of living. In plain terms, it is the 'work smarter' aspect of economic growth.

Innovation is tied closely to competition.[10] Competition in a free market is a somewhat Darwinian concept that is supposed to keep businesses honest and aligned with consumer interests. Businesses that are out of touch with the needs of their consumers tend to fail and be replaced by better-performing competitors. Since most of the well-developed economies in the world are market economies, it is often assumed that competition is necessary for economic performance. Competition, however, comes at a cost. Schumpeter aptly describes one such cost as 'creative destruction'.[11] By introducing improved products and manufacturing methods, the value of earlier investments in outmoded products and now-redundant facilities are destroyed.

A further cost of competition are invention races. It is common in the startup culture of the ICT sector today for multiple new ventures to invest in the same trend or solution to a problem, only to drop out of the race once there is a clear winner. The others who did not win this race fail to deliver return to their investors. This issue is especially apparent in patent races, where the first inventor to be granted a patent wins, leaving the competitors in the dust with almost finished inventions that are now mostly worthless due to the monopolistic IP protection afforded to the first to register a patent. However, competitive markets provide high-quality products at low prices to consumers. The benefits clearly outweigh the costs. Competition appears to be a 'least-bad' approach, one that makes businesses

[8] Jutt Bolt, R. Inklaar, H. De Jong, and J. L. Van Zanden, 'Rebasing "Maddison": New Income Comparisons and the Shape of Long-Run Economic Development' GGDC Research Memorandum 174 (2018).

[9] OECD and Eurostat, *Oslo Manual 2018: Guidelines for Collecting, Reporting and Using Data on Innovation, 4th Edition, The Measurement of Scientific, Technological and Innovation Activities* (Luxembourg: OECD Publishing, 2018) <https://doi.org/10.1787/9789264304604-en>.

[10] Philippe Aghion, Nick Bloom, Richard Blundell, Rachel Griffith, et al, 'Competition and Innovation: An Inverted-U Relationship' (May 2005) 120(2) *The Quarterly Journal of Economics* 701–28, doi:10.1093/qje/120.2.701.

[11] Joseph A Schumpeter, *Capitalism, Socialism and Democracy* (London: Routledge, 2010).

work for the consumer at an acceptable cost. This is reminiscent of Churchill who described democracy as 'the worst form of government, except for all the others'.[12]

In every economy, regardless of the type of government, competitive and co-operative processes coexist. Competing businesses may cooperate on standards development. Public goods, like education, may be cooperatively provisioned by the state in a centralised fashion. Decentralised cooperative production has, however, historically represented a negligible segment of the economy.

Such collaboration has always existed in the form of neighbourly help or bar-raising.[13] It has generally been limited in scope by proximity and shared interest. Today, the Internet enables global collaboration on Open Source development and proximity is no longer geographically restrained, which in turn allows interested parties to collaborate globally.

The beginning of the Open Source ecosystem and the development of the Internet coincided. Together they triggered advances in collaboration techniques like wikis, issue trackers, and revision control systems that enabled widely dispersed groups to work together.

Open Source offers an alternative to market competition that enables participants to collaborate where they do not plan to differentiate.

Differentiation is businesses' understanding or belief of what product features convince consumers to choose their products over those of their competitors. While those differentiating features are developed in-house and usually kept proprietary, there is no business reason to invest in the development of non-differentiating functionality individually, duplicating efforts of competitors. For example, every computing device needs an operating system, which is a complex and crucial piece of software. Consumers, however, almost never interact directly with it and are usually indifferent as to which operating system their device uses. While in the 1990's it was still common that every printer manufacturer developed their own firmware, today almost all of them are based on Linux. The same logic applies to the foundational software stack in general that is used to build consumer-focusing applications.

It is assumed today that devices contain over 80 per cent Open Source software[14] with the remainder being proprietary, differentiating code, which has been called the *Pareto Principle of Software*.

This trend comes with a drawback. To be able to build competitive products, a business *must*, in addition to the use of its own differentiating code, use the available Open Source software stack of non-differentiating software to the fullest extent possible, since its competitors will do so, and otherwise undercut their cost. As Open Source reduces research and development (R&D) cost, with such costs being shared across the creators of the code, this is factored into product prices today.

[12] <https://winstonchurchill.org/resources/quotes/the-worst-form-of-government/> accessed 11 March 2021.

[13] <http://amishamerica.com/what-happens-at-an-amish-barn-raising/> accessed 11 March 2021.

[14] 'Vulnerabilities in the Core: Preliminary Report and Census II of Open Source Software' (The Linux Foundation & The Laboratory for Innovation Science at Harvard, 2020).

As a consequence of this 'differentiate or collaborate!' has become a mantra in today's ICT industry, as the guide to decision-making for competitive, differentiating product features.

15.6 Joint Stewardship and Governance

By integrating Open Source, producers acquire crucial functionality for their devices at the expense of a partial loss of control over the software functionality of their products. Much of Open Source is developed in an open innovation model as opposed to the traditional confidential corporate R&D models.[15] As a consequence of ongoing collaboration by the development community, incremental changes to the Open Source modules are routinely and continuously shared between the participants, usually in the form of commits to a version control system such as Git.[16] Users benefit from gaining access to the aggregated contributions of other participants, the value of which commonly outweighs the cost of their own contributions. For industry contributors, participating in Open Source development is primarily an approach to save costs and increase speed of innovation by pooling R&D resources in cooperation with other participants.

One consequence of this collaborative innovation approach is that the developed Open Source functionality is available to everybody, also including non-participating parties. It is also available to contributors' competitors as others cannot be restricted from using the same code and building upon it or modifying it. By attracting contributions from many diverse stakeholders, Open Source can be very innovative and represent the current state of the art in a specific field. At the same time, it is considered a commodity in that the functionality it provides loses its differentiating value and becomes generally available to the whole world.

The combination of innovativeness, commodity character, and being a public good makes Open Source rather unique. It drives the innovativeness of the wider Open Source community and explains why Open Source development contributes positively to the common good.

It is possible to use Open Source without ever participating in its development. Indeed, an overwhelming number of users fall into this category. Passive consumers, however, will be unable to influence the technical development of the software beyond making feature requests in communities. This exposes them to business continuity risks. As the development of the software continues, it may deteriorate in quality or deviate from the functionality needs of the passive consumer's application.

[15] Henry Chesbrough, Wim Vanhaverbeke, and Joel West, *Open Innovation: Researching a New Paradigm* (Oxford: Oxford University Press on Demand, 2006).

[16] <https://git-scm.com/> accessed 11 March 2021.

Businesses consuming Open Source therefore have a stake in the viability of the ongoing development of the Open Source components they use. By engaging in the community and investing in shared development efforts for that software, businesses ensure that the Open Source products they consume match their functionality and quality requirements and, importantly, that it will continue to be maintained and secure. Invested consumers of Open Source often become contributors to it in the long term. The process is a cycle where users frequently begin to participate in both the creation of the software as well as in the governance of the community that develops it as a way actively to manage their own business risk of growing dependence on the Open Source components they use.

15.7 Contributions, Copyright, and Participation

Participants contribute code to Open Source projects which is the subject of copyright. The resulting releases of the packages usually contain contributions from many different contributors. Each release builds upon the earlier versions as a derivative work. Since the contributor base of Open Source projects fluctuates over time, the set of rights holders changes with every new release. Some contributors only submit a single patch that fixes a bug they discovered, others participate long term and may even evolve into core developers or maintainers of projects. As all Open Source licences guarantee the same minimum set of freedoms to all users, the question of who owns the code in an Open Source project may be more relevant for the individual reputation of the contributors than to the adoption or the value of the software. Many Open Source developers care strongly about being properly attributed for their work as this builds their reputation and their personal value, despite there being minimal restrictions on the distribution of the software they created.

The contributors who currently develop and maintain an Open Source project are typically a subset of all of the copyright holders. The contributions of their predecessors facilitate the continued development of the software and the prior participants' licence decision will impact release of new versions. The current group of contributors assumes *joint stewardship* over the technical development of the project and the management of the community (whether or not they are joint owners).

To become an active stakeholder in an Open Source project, newcomers must engage with the community that develops it. Since contributor fluctuation is quite normal in Open Source communities, a key aspect of community governance is to ensure continuity in the event of changing participants. In communities that have been active for a period of time, the currently active stakeholders steer the project on behalf of themselves and the previous contributors who are likely copyright holders on the code, but who no longer participate in the project. In some organisations, being an active contributor or qualifying for membership in the technical or administrative steering bodies is tied to a financial contribution or organisational membership fee. These fees are nominal in communities but may be quite substantial in foundations and may depend on the size or turnover of the contributing organisations. They cover administrative, governance, and community management costs. The software the community produces is still Open Source and free to use, irrespective of such fees. Governance and Open Source licensing are two separate dimensions.

15.8 Communities, Contributors, and Merit

The term *community* is crucial to Open Source but can confuse as it is used with different connotations. To understand how Open Source is produced, it is necessary to distinguish Open Source communities from other organisations and to derive the specific functions performed by communities from this. Some people intuitively assume that Open Source communities consist primarily of enthusiastic volunteers collaborating for the common good. This is, however, not necessarily the case, and increasingly businesses and paid developers make up significant proportions of community.

The mix of different types of contributors like individual volunteers, businesses, or community staff is referred to as *community composition*. Quantitatively, most contributions originate from businesses and the majority of individual contributors participate in Open Source development as part of their employment,[17] making the communities *hybrid* or *heterogenous* in their composition. While licensing defines whether a piece of code is Open Source or not, the *openness* of a community is defined by its governance norms. Communities with many diverse participants typically prefer open, transparent, and accountable governance processes.

Producing Open Source is of course a necessary aspect of any Open Source community, but not sufficient for a definition. Different entities, state authorities, or anonymous donors may release code under Open Source licences without ever engaging with others to build a community.

Open Source communities differ in *how* their products are created, which is a question of governance. Especially volunteer-driven communities care strongly about the values they communicate to their potential participants. The KDE Manifesto, for example, lists open governance, free software, inclusivity, innovation, common ownership, and end-user focus as essentials.[18]

[17] D. Riehle, P. Riemer, C. Kolassa and M. Schmidt, 'Paid vs. Volunteer Work in Open Source,' *2014 47th Hawaii International Conference on System Sciences*, Waikoloa, HI (2014), 3286–95, doi: 10.1109/HICSS.2014.407.

[18] <https://manifesto.kde.org/> accessed 11 March 2021.

A *contributor* is an individual or organisation that invests resources in the creation of a community product. If a contributing developer is employed by an organisation, the employer pays the salary and may gain the copyright on the contributed code. In many organisations individual developers are allowed to contribute in their own name. This creates a lack of clarity as to whether an individual contributes on their own account, that of an employer, or one of a number of organisations with whom they are associated.

In the case of doubt, from a governance perspective the entity to be considered the contributor should be the one that has the authority to decide what effort to contribute and to what product or community. Depending on the internal arrangement, this may be the individual or the company.

There are many different types of possible *contributions*, not just the provision of code. Other examples are organising a community event, translating software to local languages, maintaining the community web site or writing newsletters. The different types can be normalised to contributions of time, money, or knowledge in the form of experience or expertise. By participating, contributors gain *merit* in the community which defines their standing within their peer group. Merit develops organically based, for example, on technical expertise, long-term commitment, or the social role and reputation of the individual. Merit or the value of contributions are at times difficult to measure in quantitative terms.

While Open Source licences govern the contributor–user relationship, governance structures govern the contributor–community relationship.

Contributors participate in Open Source development and engage with the community voluntarily and based on their own desires. Even though anybody can use, study, modify, and redistribute Open Source, nobody can be forced or coerced to contribute to it, and arguably there is no moral imperative to do so. Any individual or organisation that contributes decides that it is the right thing to do for them.[19] Engagement in Open Source communities may be explained based on Hirschman's concept of exit, voice, and loyalty. Hirschman researched consumer loyalty based on 'a conceptual ultimatum that confronts consumers in the face of deteriorating quality of goods: either "exit" or "voice" '.[20] Long-term consumers of a product develop loyalty to it in that they would rather continue to use it and do not wish to change to a different one. When applying Hirschman's concept to Open Source, the change of quality is that of the governance of the community, while the loyalty is that of a long-time participant. The option of having a voice in Open Source comes from engaging in the community to maintain its quality and to participate in joint stewardship. The exit option is to stop participating and to

[19] Josh Lerner and Jean Tirole, 'The Economics of Technology Sharing: Open Source and Beyond' (2005) 19(2) *Journal of Economic Perspectives* 99–120.

[20] Albert O. Hirschman, *Exit, Voice, and Loyalty: Responses to Decline in Firms, Organizations, and States* (Cambridge, MA: Harvard University Press, 1970).

disengage from the community. Over time, contributors tend to feel very strongly about their communities and develop a sense of belonging. When considering whether or not to continue to participate, they often prefer not to let their fellow contributors down.

A specific form of exit in Open Source occurs where there is a fork. A fork is a split of the development community where a group of contributors establishes a new 'centre of development'[21] to continue the development of their own separate version of a product. Well-known forks are the LibreOffice/OpenOffice Elasticsearch/Opensearch and the OwnCloud/Nextcloud splits. Forks are a corrective measure that ensures that community governance stays aligned with the interests of the currently active contributors.

They are made possible by the essential provisions of Open Source licences and almost always represent an issue with community governance, illustrating further the duality of Open Source licensing and community governance as separate dimensions. Forks come at a cost, for example in the form of a split of the contributor base, added technical complexity, or interoperability issues, which is why contributors do not take this step lightly.

This fact focuses the community on whether there is a less destructive way of tackling the issue which instigated the possibility of forking in the first place. Accordingly, the threat of forking provides additional checks and balances over how the community governs itself.

Voluntary participation together with the *potential of forks* keep contributor interests and community governance in line.

The combination of licensing and governance provides a suitable definition of what makes an Open Source community: An *Open Source community* 'is … a social group of contributors that participate voluntarily in the production of public information goods'.[22] The two functions that communities need to provide based on this definition are:

- *community governance*, which determines the perceived quality of the organisation in the eyes of the contributors and influences their voice-or-exit decisions; and
- *community management* as the task to motivate contributors to join, actively participate in the community, and to stay active instead of exiting.

Contributors join communities if they expect to achieve their goals more easily as part of the group compared to working alone. This comes as a trade-off, as to become a part of the community, a share of the contributor's investment needs to

[21] <https://opensource.com/article/19/1/forking-good> accessed 11 March 2021.

[22] Böhm, 'The Emergence of Governance Norms in Volunteer Driven Open Source Communities', see note 7.

be directed towards being a community member as opposed to the development effort.

The term *governance* refers to all of the processes of social organisation and coordination within the group. Essential aspects of community governance include the explicit and implicit organisational structure, the decision-making and conflict resolution processes, and the social order of the community.

The reasons why communities experience contributor fluctuation can be conceptually separated into changes in motives and changes in motivation. The motives of a contributor may change based on external developments. People may graduate from university where they enjoyed Open Source development, or start a family and intend to spend more time with their kids, or change jobs and now work on different things. Communities need to accept and possibly even encourage such changes as a sign of a healthy personal development. Changes in motivation, however, are caused by internal community processes that affect organisational quality as perceived by the participant. They are determined by the community's governance norms and maintained through community management. How the community makes decisions and resolves conflicts, the impact of speaking up to influence the group, how decisions are enforced in the face of voluntary participation, the support the community provides to the creative development process, and the delineation of community members and outsiders all influence the perceived quality of the organisation. Contributors participate voluntarily in Open Source communities. They engage in the production as well as the governance processes. Communities facilitate contributions through their governance and attract contributors through community management.

15.9 Value at the Edge of the Commons

At the intersection of Open Source and business there is an apparent *tragedy of the commons.*[23] Developing Open Source is a virtuous effort that contributes positively to society by improving the available technology stock. At the individual level, contributors are passionate about their work and love what they do. On the other hand, many businesses that aim to create value by developing and building upon Open Source technologies struggle to find viable business models. Some have questioned the sustainability of the Open Source development model as a whole even though the wider Open Source community is thriving.

One source of confusion in this context is the much-repeated question of 'how to make money with Open Source'. This question is difficult to answer because it reduces benefiting from Open Source to capturing value through the generation

[23] Garett Hardin, 'The Tragedy of the Commons' (December 1968) 162(3859) *Science* 1243–8.

of revenue which is explored more fully at Chapter 16. It has long been established that there is no special Open Source economy.[24] Instead, the production of Open Source follows basic economic principles in a process that can be explained by breaking down more systematically how businesses benefit from Open Source.

15.9.1 The global upstream/downstream network

To illustrate how to position businesses in the Open Source value chain, it is necessary first to look at how the wider Open Source community organises itself. Individual communities develop specific Open Source products representing parts that need to be integrated into consumer-focused software and hardware products in order to be useful. The mechanism that coordinates the efforts of the various specialised communities is called the global upstream/downstream network. This network integrates the work of the various specialised communities into a technology stack suitable for end-users or as platforms for commercial products. This upstream/downstream model of collaboration within the wider Open Source community uses the mental image of a large river that collects the water from many tributaries (the communities) and delivers it to the ocean (the users).

No central decision-making body exists to coordinate within the global upstream/downstream network; instead, the communities operate autonomously and react to the stimulus from feedback and contributions in a competition for relevance and adoption of their solutions. Product improvements originate in the communities and are integrated 'down the stream' by more and more complex aggregated products. Feedback such as bug reports and requests for improvements, but also patches meant for integration into the upstream projects, are generated closer to the users, and then travel 'up the stream' to be eventually integrated by the originating community for that package. Practically all relevant Open Source solutions are part of this network that coordinates between supply and demand of Open Source contributions, resulting in complex, highly integrated products, for example Linux distributions or device platforms like Android or Yocto.

All copyright licensing within the wider Open Source community is automatic and transitive. It is automatic in that no negotiation takes place between the authors and the users of the software, and use of the software is subject to the terms on which is it is licensed. Licensing is transitive in that everybody in possession of the software is able to redistribute it to make it available to any other party without the need to refer back to the original author. By way of the combination of automatic and transitive licensing, the wider Open Source community avoids potential 'anti-commons' situations. In an anti-commons situation, the effort to acquire all

[24] Steven Weber, *The Success of Open Source* (Cambridge, MA: Harvard University Press, 2004).

necessary IP becomes prohibitively high, resulting in an underuse of available assets.[25] The widespread reuse and integration of Open Source solutions in the upstream/downstream network potentially results in an exponential increase in the number of licensing relationships that can easily produce an anti-commons situation. It is therefore essential for the functioning of the global upstream/downstream network that all necessary rights are acquired *ex ante* and without the need for negotiation. This is one reason why patent holders have found it difficult to combine the use of Open Source with revenue-bearing patent licensing. To avoid the necessary negotiations with patent holders after the software has already been used, the Open Source community tends not to adopt patent encumbered technologies.[26]

15.10 Open Source-Related Products and Service

Businesses operating in the Open Source ecosystem offer a combination of *goods* and *services*. These terms are loaded with different meanings, for example based on whether what is sold by the business is a unit of a good or billable hours. To avoid confusion, the distinction made in our context is that offering a product requires the right to do so, for example based on the ownership of physical goods or the rights to sell commercial software licences or redistribute code, while services can be sold in a way that they complement a good somebody else possesses.

Applied to Open Source, businesses must make a choice regarding how to manage IP related to the value proposition they offer to consumers. To pursue product-based business strategies, they must retain appropriate rights over the complete product source code, for example through the application of contributor licence agreements (CLAs) as discussed in Chapter 4. These agreements ensure that the business has the necessary rights to sell proprietary licences to the software, at the expense of creating asymmetry between the contributors to the software: While those participants that submit improvements and bug fixes under the CLA can contribute to and use the software as Open Source, only one entity has the rights to benefit commercially from it. Such 'single-vendor' business models require a strong market position usually based on thought leadership and innovativeness that convinces external contributors to participate.[27] MySQL, Qt, or Asterisk are examples of products that have been successfully developed under such a goods model. In contrast, offering services related to Open Source products does not

[25] Michael A Heller, 'The Tragedy of the Anticommons: Property in the Transition from Marx to Markets' (1998) *Harvard Law Review* (1998): 621–88.

[26] Knut Blind and Mirko Böhm, *The Relationship Between Open Source Software and Standard Setting* in Nikolaus Thumm (ed) (Brussels: Publications Office of the European Union, 2019).

[27] Dirk Riehle, 'The Single-Vendor Commercial Open Course Business Model' (November 2010) *Information Systems and E-Business Management* 1–13.

require appropriation of the software by the vendor of the service. Anybody with the necessary expertise may offer support, custom development, or operate Open Source-based solutions for clients without the need to own the copyright to the original code and companies may offer support for code that is distributed by others.

Using this distinction between goods and services as the basis for revenue generation, Open Source-based value propositions can be further broken down based on their function in the value chain. Software products may be used as foundational technology or as consumer technology.

Foundational technologies are the intermediate products or building blocks of the tech sector that provide common functionality from the operating system to web-based communication or user interface frameworks. They are combined by manufacturers into more consumer-ready devices, even though the consumers often do not know or necessarily care which exact software the device contains so long as it does the job.

Consumer products are made to satisfy concrete needs rather than for reuse or as means of production. Open Source solutions such as operating systems, a boot loader to prepare the device and start it running, programing language runtimes or databases are common building blocks used in many devices that consumers expect but are usually indifferent to. Even most proprietary software regularly also includes common Open Source modules for these.

Businesses may offer either *vertical integration* where they operate Open Source as a service for their customers or offer services that are *complementary* to the product itself.

Building custom websites based on an Open Source web framework vertically integrates the framework into a higher-level application. Similarly, a cloud provider that provides instances of an Open Source database vertically integrates the software into their main product, the operation of data centres. To provide vertical integration, the service vendor requires expertise on how to use the underlying software modules and domain knowledge about the intended application, but—unless the vendor wishes to extend their functionality—not necessarily knowledge on how to develop the software. Vertical integration creates value in the eye of the consumer on top of the underlying Open Source solutions.

Complementary services in comparison focus on supporting the Open Source modules themselves, as in custom feature development or long-term maintenance of a library. Instead of building higher-level functionality, they make foundational technologies viable for inclusion into other products or for use in different domains. Since the Open Source ecosystem is built upon the idea that everybody can maintain and extend the software, complementary services are an essential element of it. Many Open Source developers make a living from being the main contributors to software modules and getting hired to improve or extend them. Such services are not free. The philosophy that the software should be free while it is at least partially developed commercially is an inherent idea of Open Source.

Even selling copies of the software is explicitly allowed by the terms of Open Source licences, which is represented by the slogan that 'Open Source is commercial'.[28] Instead, the Open Source community distinguishes between Open Source and proprietary software based on the licences applied.

Vertical integration requires a higher grade of domain knowledge, while horizontal complementary services require more technical expertise about how a specific software module is implemented. The vendors involved similarly develop into different roles. Vertical integrators often regard the Open Source modules used as stable and complete and focus on providing them to a wide range of customers. Horizontal service vendors care about the software as something to maintain and improve. They contribute more changes to the software compared to the vertical integrators, sometimes leading to criticism that those only use the software without 'giving back to the community'. In particular, businesses that pursue growth strategies based on Open Source technologies, sometimes funded by venture capital, find it difficult to convert vertical integrators into paying customers. Since there is no obligation to contribute back or to 'pay a fair share', these difficulties illustrate problems in the underlying business models rather than with the sustainability of Open Source development. To be able to offer convincing value propositions to potential customers, businesses need to maintain a level of control over the goods they are selling. Open Source in itself as a public good does not provide this leverage.

Businesses have tried to combine Open Source with various approaches to capture its value. The single-vendor model mentioned previously is one of them. Others include *trademark* licensing programs, the acquisition of *patents* parallel to software development, or attempts to control the market by enforcing the use of standards covered by SEPs.[29] All of these approaches represent a trade-off between software freedom and capturing value in a business.

In summary, the Open Source-based value propositions can be broken down into:

- foundational versus consumer-oriented products; and
- vertically integrated versus horizontally complementary services.

The vast majority of well-known businesses in Open Source sell services, a fact sometimes obscured by services being marketed in a way that is similar to physical goods.

The subscriptions to technical support that Red Hat offers are a service complementary to the Linux distribution that contains thousands of software package

[28] <https://www.gnu.org/philosophy/selling.html> accessed 11 March 2021.

[29] Björn Lundell, Jonas Gamalielsson, and Andrew Katz, 'On Implementation of Open Standards in Software: To What Extent Can ISO Standards Be Implemented in Open Source Software?' (2015) 13(1) *International Journal of Standardization Research* 47–73.

developed by the wider Open Source community which Red Hat does not need to own. GitLab (the company) offers hosted software development infrastructure as a vertically integrated or support for on-premise installations as a complementary service based on GitLab (the software). GitHub hosts many important Open Source projects, even though it offers a vertically integrated service that itself is not free software. Android phones are proprietary consumer products based on an Open Source foundation, the Android Open Source Project.[30]

15.11 The Benefits of Open Source in a Business Context

The success of Open Source has incentivised entrepreneurs to build revenue streams based on it, as is discussed in detail at Chapter 16. However, 'making money with Open Source' is only one way a business can benefit from Open Source. With regards to business strategy, there are three possible scenarios regarding how Open Source can be useful to businesses:

- Open Source can be useful without the goal of direct financial benefit, or
- it can be used to directly to *generate* revenue by being sold, or
- it may provide a way to *reduce* the cost of a product.

All three scenarios are relevant in practice. Many software engineering tools like compilers, build systems or programing language environments today are developed in an Open Source-first approach. These tools enable ecosystems of specialised functionality and drive developer engagement through knowledge transfer. By standardising their ICT infrastructure and engineering toolchains, through Open Source, a business can significantly reduce up-front expenditures and focus the R&D budget on consumer-relevant product functionality which it believes drives consumer choice. Similarly, Open Source enables interoperability, for example through shared application programing interfaces (APIs), which allows independent organisations to build solutions that integrate with each other.

A different approach is the loss-leader - software products like web browsers that are distributed as Open Source to attract consumers and market other value propositions. All these approaches benefit businesses indirectly without generating revenue for them.

To build revenue streams, businesses can offer products or goods for licensing or use them to offer integrated or complementary services.

In the single-vendor case, all rights to the source code are controlled by the vendor, enabling them to license the same source code to different customers

[30] <https://source.android.com/> accessed 11 March 2021.

under different terms. This dual-licensing or multi-licensing approach represents a price differentiation mechanism that allows vendors to achieve a larger market share through the additional adoption of those consumers who choose the Open Source solution out of a preference for the licence or to save cost. The product is typically offered under a copyleft Open Source licence that requires customers to disclose their own source code, with the alternative offer to buy commercial licences to avoid this requirement so promoting the uptake of commercial licences to avoid concerns, with respect to the use and impact of such copyleft code.

Because the necessary CLAs create asymmetry between the contributors, this approach is controversial in Open Source communities, as was seen in January 2021, when Elastic moved two of its products from Apache 2.0 to the proprietary SSPL licence and was able to do so only because it had received CLAs from its community of contributors.

For a company to achieve a reputation in the market that enables it to apply multi-licensing approaches requires thought leadership and innovation to build the necessary goodwill with both contributors and consumers. Few are successful and failing companies risk to losing external contributions and may end up carrying the development cost of the complete product. There is also a risk of forks, which was the consequence in the Elastic situation.

A third benefit is cost reduction in building software or devices. In the simplest case, substituting a proprietary software module with an equivalent Open Source implementation eliminates the licensing effort and importantly cost or royalty.

More commonly, the required functionality is needed by many companies but not readily available in the market. This incentivises businesses to pool R&D cost with others that have the same needs, effectively reducing their own investment to a fraction of the overall cost. In such a set-up, participants generally expect to be equals amongst the other contributors. Open Source licensing facilitates the collaboration. Projects are commonly set up as not-for-profit organisations or at Open Source foundations and typically industry-driven as is discussed further in Chapter 18.

The same company may develop different parts of its product portfolio under different models. It is quite common that manufacturers compete in the same market segment with their products while at the same time collaborating in Open Source projects. Since the Open Source product itself is always free to all interested users and for all purposes, a 'good business model is simply one that succeeds in creating additional value at the edge of the commons'.[31] One essential question for every Open Source-related strategy is how the business benefits from its participation in the Open Source ecosystem by one of these three approaches:

[31] Weber, *The Success of Open Source*, see note 24.

- by generating revenue; or
- by reducing cost, or
- by realising other non-financial benefits.

15.12 Differentiating in the Eyes of the Consumer

Businesses decide where to compete and where to collaborate based on what they expect to be differentiating product features that convince consumers to prefer their products over those of their competitors. Common product features are best implemented using existing Open Source solutions both to share R&D cost with other contributors and to benefit from the joint expertise of the stakeholders involved. The differentiating product features are more commonly developed in-house by the vendors themselves and not Open Source. Embedded or mobile devices today share a large part of their foundational software modules, while they implement user experience and application-specific business logic elements as proprietary software.

This differentiation is exclusively in the eyes of the consumer. There are two common logical fallacies:

- First, contributors assume that because they invested time and resources into developing a product, they are entitled to compensation. Unfortunately, it is quite common that businesses make the wrong bet and develop products that are not convincing to consumers. In an open economy, the 'fair compensation' a business should expect as the return on R&D investments is the value the market assigns to the product. Only by focusing investments towards those product features that consumers value can a business be successful in the market.
- Secondly, Open Source developers sometimes expect that because *they* have contributed valuable code, consumers should work with *them* and hire them for example for ongoing development or operational support. There is, however, no intrinsic value for the consumer or a vertical integrator in a business relationship with the core developers of a software unless this relationship benefits both sides beyond the free licence to the software, for example by adding value through knowledge transfer.

Developers and businesses that produce Open Source solutions must find ways to differentiate their value propositions. Positive differentiators include:

- the perception of quality and innovativeness based on the joint expertise of different stakeholders who participate in its development;

- enhanced trust based on the ability to verify the functionality and integrity of the software;
- the sustainability of the development model;
- reduced lock-in; and
- the option to procure maintenance and feature development work from a variety of providers and other factors that promise that the software is useful to the consumer in the long term.

Many of these positive connotations that Open Source vendors can utilise to differentiate are influenced by the impression of community health, which is commonly assessed with metrics like the number of independent organisations and individuals participating in the Open Source development process or the overall number of contributions raised by the community. These metrics reflect negatively on single-vendor models and partially explain the hesitation especially of vertical integrators to engage in a business relationship.

While the assumption that corporate Open Source users in particular should return a 'fair share' to its developers is understandable, there is no imperative to contribute from the individualistic economic perspective (it could be explained by other disciplines).

However, conflicts based on free-riding behaviour are rare in the Open Source ecosystem, and almost all involve Open Source vendors that implement the single-vendor model.

Some Open Source vendors, particularly those with venture capital (VC) funding criticise their users and the community for not giving enough back to them. This 'community bashing' is reminiscent of politicians that criticise their electorate for exercising their free will not to support their policies. More rationally, the behaviour of the consumers can be explained by the negative differentiation effects that is caused by the reintroduced vendor lock-in or the lack of sustainability of the software development process caused by the absence of a healthy, diverse developer community.

In short, some single-vendor Open Source businesses attempt to reintroduce proprietary software development models and strong vendor lock-in based on an Open Source product, which reduces software freedom. These attempts contribute to the negative reputation of contributor licence agreements that the vendors require to pursue these strategies.

15.13 The Role of the Volunteer Community

In the discussion of the economics of Open Source, the focus is less on the volunteer community since it contributes only a fraction of the overall Open Source development effort and is less engaged in the larger Open Source foundations and the discussion of commercial models. The volunteer community is, however, a core element of the wider Open Source ecosystem. This is illustrated by the early success and market adoption of Open Source that happened when business participation in community development was still considered an extravagance. The volunteer community acts as a driver that stimulates Open Source innovativeness.

The individual motivation of volunteers to participate in Open Source development differs from the business rationale outlined earlier. Developers start contributing to Open Source projects based on their own interests and technical needs and over time evolve a deeper engagement with the developer community to gain a sense of achievement and belonging. Open Source culture builds on the careful, transparent, and consensus-focused governance that the communities set up based on the paradigm of voluntary participation and the correctives affected by the possibility of forks. Many contributors consider Open Source development something worth fighting for. They focus more heavily on the virtuosity of contributing and the benefits of Open Source to society. Software freedom has more importance to them than the availability of the source code. As such, the engagement of the decentralised Open Source developer community provides an important safeguard of software freedom and represents the interests of civil society.[32]

From the perspective of participating individuals, the positive freedom to use, study, modify, and redistribute the software gains more emphasis compared to the absence of constraints. This perspective has defined the debate on openness and freedom indicative in the histories of the FSF and the OSI as opposed to the industry-led Open Source foundations. Representation of the decentralised volunteer community of Open Source contributors often focuses on charitable goals and is separate from industry-led foundations which are essentially trade associations. This makes volunteer community organisations natural counter-parts for policy-makers and gives them a sometimes oversized credibility and reputation as trusted advisers. Compared to the industry associations, volunteer organisations operate on small budgets and staff. There is a generally fluid transition from being a volunteer contributor to a corporate one upon graduation or with the creation of start-up businesses, emphasising the importance of Open Source as a knowledge transfer mechanism. The volunteer community is an essential and necessary part of the wider Open Source ecosystem that supports the competitiveness of the software market and the alignment of technical innovation with the interests of society.

[32] Böhm, 'The Emergence of Governance Norms in Volunteer Driven Open Source Communities', see note 7.

15.14 Competition in the Wider Open Source Community

As there is generally no direct remuneration for Open Source contributions, competition in the Open Source ecosystem is not about revenue or market share. Contributions are not market transactions in which two parties negotiate a trade they assume to be of similar value for both sides. However, the Open Source community exhibits a fast pace of innovation, develops new state-of-the-art technology, and swiftly reacts to changes in the technology needs of the consumers. There is competition within the communities, between the communities in the Open Source ecosystem and with the rest of the market and even government.

Within the communities, contributors compete for the integration of their code to be released with the community products. Since the motivation of individual contributors is often driven by non-monetary factors like a sense of achievement, positive creativity, or pride, the effort that is invested into an incremental improvement of a specific feature is at times higher than justified purely by technical requirements. This perfectionist attitude of 'it is done when it is done' enhances overall product quality. The prestige of proven contributions to important Open Source projects or a good reputation as a contributor is valuable enough that in the software sector, they come to be considered a part of the developer CV. This combined with the potential global participation of individual developers makes for a rather fierce intra-community competition.[33]

Open Source communities compete with each other for the adoption and integration of their solutions within the global upstream/downstream network and eventually the consumer market. This mindset drives the acquisition of new contributors and the continued development of the software that defines the relevance of the community and its ability to facilitate contributions and raise funding. Participants in Open Source communities bet on the adoption of their community's software to help them realise the benefits from their contributions. This inter-community competition for relevance and adoption causes swift technological cycles that displace even well-known solutions with a large contributor base once a more promising alternative emerges, as illustrated by the competition between the OpenStack and Kubernetes communities. The participants in these communities frequently stay involved and continue to contribute to Open Source, however they quickly shift attention and contributions to the newly dominant solutions. While individual communities grow and shrink, the overall community of Open Source participants seems to mostly grow slowly and steadily. The upstream/downstream network exhibits powerful positive externalities of community health

[33] Y. Yu, G. Yin, H. Wang, and T. Wang 'Exploring the Patterns of Social Behavior in GitHub' in *Proceedings of the 1st International Workshop on Crowd-Based Software Development Methods and Technologies, CrowdSoft 2014*, 31–36.

and size and fast adoption of new technologies similar to the tipping markets of Internet products.[34]

When Open Source products substitute proprietary products, the community competes with private enterprise. The inherent freedoms and public good character of Open Source make it difficult for businesses to compete since they need to offer strong value propositions to justify a non-zero price. This kind of one-to-one competition between free and commercial products was however more common in the earlier days of Open Source adoption as a way to challenge the market position of entrenched incumbent software vendors, resulting, for example, in LibreOffice, the Linux kernel, and Apache. Collaborating on Open Source solutions challenged the market position of the incumbents and forced their products to be more consumer-oriented, without necessarily replacing them. Markets for some proprietary products have practically disappeared, such as those for commercial software development tools or proprietary embedded operating systems. In these cases, collaboratively developed products perform better based on a wide stakeholder participation in the development process. Today, especially industry participants have adopted a conceptual separation between a competitive zone where consumer-oriented, differentiating product features are developed and a collaborative zone that creates the underlying non-differentiating functionality. Different behaviour is expected in these zones. The competitive zone covers a smaller part of the overall software of a device or application and functions mostly unchanged in terms of development processes and IPR management compared to traditional R&D. In the collaborative zone, a key principle is joint stewardship over the community products with the expectation of *ex ante* licensing of all relevant IP. Based on this collaborative approach, businesses engage in a model of 'Continuous Non-Differentiating Cooperation'.[35]

An under-researched aspect of Open Source community collaboration is the way it competes with state actors. Open Source offers technical solutions that may challenge established political processes, as for example in e-democracy applications, and facilitates new forms of political participation. The production of private goods is coordinated primarily within firms or between firms in markets, while public goods are mostly provisioned by the state.[36] Open Source enables a 'fourth transactional framework'[37] that provisions public goods in an alternative, decentralised fashion.

The cross-border collaboration of Open Source communities also challenges established policy frameworks, which is of particular relevance with Brexit and other

[34] Shapiro and Varian, *Information Rules*, see note 2.

[35] Blind and Böhm, 'The Relationship Between Open Source Software and Standard Setting', see note 26.

[36] Ronald H Coase, 'The Nature of the Firm' (November 1937) new series 4(16) *Economica* 386–405.

[37] Yochai Benkler, 'Coase's Penguin, or, Linux and "the Nature of the Firm" ' (December 2002) 112(3) *The Yale Law Journal* 369+.

geo-political shifts in the US, China, and Europe, in particular calls for digital sovereignty. However, local legislation like the European General Data Protection Regulation (GDPR) and governmental policies such as the Chinese 'Great Firewall' still have significant impact. There is not much research on the potential competition between the Open Source community and government at this juncture. This may, however, become an important field of inquiry for both developing and developed countries, especially as the impact of Open Source correlates with policy objectives like the United Nations Sustainable Development Goals.[38]

Overall, the wider Open Source community represents a very competitive environment that results in fast-paced technical innovation, reduced barriers to entry, and a challenge to incumbent market positions. The introduction of Open Source both from a licensing and a collaboration perspective usually increases competition by providing alternative models and approaches. Open Source community governance norms are, however, not fully standardised, with a theoretical possibility that participants may form market-controlling clubs and it is normal to have antitrust or competition policies to avoid this. The public good character of Open Source combined with open, transparent governance norms generally inhibit possible anticompetitive behaviour. A key contribution of Open Source to economic growth is the provision of baseline technologies that represent the current state of the art and are available to everyone.

15.15 Compliance, Social and Market Transactions, and Zero Price

Key effects of Open Source collaboration include:

- the reduction of transaction cost of participation and
- the reduction of barriers to entry for newcomers.

Reduced transaction cost opens the collaboration process to an overall larger constituency and in particular invites participants for whom staff cost and membership fees, for example in traditional standards development have been a challenge, namely small and medium-sized enterprises (SMEs).[39] Reduced barriers to entry for newcomers improve the chances for example of university spin-offs and start-up companies to compete with incumbents, improving competitiveness.[40] These benefits depend on efficient IP management across the whole Open Source

[38] <https://sdgs.un.org/> accessed 11 March 2021.

[39] Blind and Böhm, 'The Relationship Between Open Source Software and Standard Setting', see note 26.

[40] Frank Nagle, 'Government Technology Policy, Social Value, and National Competitiveness,' Harvard Business School Strategy Unit Working Paper, no. 19–103 (2019): 1–50.

ecosystem. An implicit expectation of *Open Source compliance* towards all participants requires adherence to the obligations from all consumed Open Source licences and is regarded as a *hygiene factor* by the wider Open Source community. Hygiene factors create dissatisfaction by their absence.[41] In the context of Open Source, compliance is considered as a tool of the trade. Uncertainty about the compliance of suppliers and consumers or necessary litigation undermine the fabric of the global upstream/downstream network by negating the reductions in transaction cost of participation and re-erecting barriers to entry.

Open Source is made freely available to everybody. A price of zero triggers a different response in actors even compared to a bargain low price. It transforms the exchange between communities and consumers from a market transaction where parties negotiate for gains at the others expense to a social exchange with an agreement on behaviour that is beneficial to everybody involved. Asking for remuneration or negotiating over commitments constitutes anti-social behaviour in a social exchange. Instead, there is an expectation of fairness and reciprocity.[42] This explains why Open Source compliance should not be managed as a risk but considered an imperative.

Reciprocal licences model Open Source usage as a social transaction by asking the consumer to act similarly to the licensor. They aim at symmetry between contributors and users and at ensuring that using the software remains a social exchange for the long term. Because it is intuitively understood that even a small number of bad actors reduce the overall willingness to engage in social exchanges, the Open Source community is sensitive to licence violations and willing to litigate against them.[43] This attitude of the wider Open Source community is in line with recent research that shows a weakening of social norms in the face of even a few bad examples.[44] Considering Open Source participation as a social exchange again helps to explain the negative reputation of CLAs since they are based on market exchanges. Contributing as an individual to a single-vendor product under CLA is more similar to paying taxes than to bringing a dish to a dinner with friends.[45] Similarly, programs that put out bounties for adding features or fixing bugs reframe contributing to Open Source as a market transaction.

A framework for social transactions is disrupted as soon as any party starts negotiating about remuneration, which is why the Open Source community avoids any form of negotiation of terms between contributor and consumer. Instead, Open Source licences are *ex ante* agreements where all rights necessary to build the

[41] F Herzberg, B Mausner, and BB Snyderman, *The Motivation to Work* (New Brunswick, US and London, UK: Transaction Publishers, 1993).

[42] Dan Ariely, Uri Gneezy, and Ernan Haruvy, 'Social Norms and the Price of Zero' (2018) 28(2) *Journal of Consumer Psychology* 180–91.

[43] <https://gpl-violations.org/> accessed 11 March 2021.

[44] Ariely, Gneezy, and Haruvy, 'Social Norms and the Price of Zero, see note 42.

[45] Benkler, 'Coase's Penguin, or, Linux and "the Nature of the Firm" ', see note 37.

desired outcome are secured from the start. The agreements are standardised into a number of approved licences that model the three basic modes of Open Source collaboration:

- strong copyleft,
- weak copyleft, and
- and permissive.

Only a small number of licences are used in recent practice and new licences are rarely approved. Combining Open Source licensing with other rights such as patents or trademarks that require *ex post* licensing of terms has generally not been successful.[46]

Maintaining licence compliance has gained considerable complexity due in particular to modern hardware devices such as general-purpose computers that include all sorts of computing, storage, user interface, and networking functionality.[47] Manufacturers are required to document the complete use of Open Source in the devices according to the obligations from the licences contained in them. Recent initiatives aim at reducing this complexity to ensure the viability of the Open Source supply chain.[48] Industry and community best practices in this context continue to evolve.[49]

15.16 Open Source as Community-Provisioned Public Good

Open Source affects society at all layers of the economy. Individuals participate for their own reasons and learn valuable skills. They find enjoyment, a sense of achievement, and belonging and opportunities to be a productive part of something bigger: the wider Open Source community.

Businesses find both threats and opportunities from Open Source. It greases competition and fosters the innovativeness of the tech sector by providing free baseline technologies, potentially weakening market positions of incumbents and causing creative destruction of existing assets. On the other hand, Open Source offers plenty of business opportunities and faster, cheaper ways to produce modern applications and devices.

For every Open Source-derived product a business intends to market, it needs to answer (at least) these three questions to describe the value proposition to the consumer:

[46] Blind and Böhm, 'The Relationship Between Open Source Software and Standard Setting', see note 26.

[47] Armijn Hemel and Shane Coughlan, *Practical GPL Compliance* (Linux Foundation, 2017).

[48] <https://www.openchainproject.org/> accessed 11 March 2021.

[49] <https://reuse.software/> accessed 11 March 2021.

- What is the revenue model of the product? Is the goal to provide a defined functionality at the lowest possible cost, or to induce indirect benefits without directly generating revenue, or is the product meant to be sold in the market to establish a revenue stream?
- What type of Open Source-related good is it? Is it a service of either the horizontal complementary or the vertically integrated kind, or is it a product which may either consumer-oriented or a foundational technology?
- What is different about what their offer? The business needs to market the product to the consumer based on differentiating product features, while including the expected but non-differentiating features at low cost in order to sell the product at competitive prices.

Based on the answers, participants can derive ways to build and market their products. Some combinations represent well-known approaches. Kubernetes and Linux are non-differentiating foundational technologies that reduce the cost to operate data centres or build devices. They are developed at Open Source foundations in a model where participants pool R&D cost. ChromeOS and clang are differentiating consumer technology that realise indirect benefits by enabling developer ecosystems or marketing complementary services. They are developed as Open Source by a single company or a small group of stakeholders. Android phones and proprietary apps in general are differentiating consumer-oriented products intended to generate revenue. Their differentiating features are developed in-house while they build on foundational technologies developed in collaboration with others.

Not all combinations of answers have been successful in the market. Building differentiating consumer-oriented products as Open Source that are supposed to directly generate revenue lacks the necessary differentiation in the eyes of the consumer. These approaches suffer from an inherent contradiction that the purpose of a business is differentiation in the market, while the essence of Open Source is to be non-differentiating. An alternative is the Red Hat model to differentiate based on services that complement the free product. By answering how they create additional value on the edge of the commons, businesses can embrace Open Source and benefit from it.

From a macro-economic perspective, Open Source is a toolbox that is part of the available technology. Software only has a tangible effect if it is executed in an application or device. Because of that, there is no possible economic downside from the development effort itself of the wider Open Source community at the macro level, while Open Source is proven to be useful and drive innovation. Hence Open Source *contributes positively* to the common good.

Where Open Source participation is regarded as a social transaction, free riding is not common. By resolving the free rider problem, Open Source development processes open up the possibility for the decentralised collaborative development of public information goods at a large scale.

Open Source benefits society by combining innovative state-of-the-art technologies that are at the same time commodities and public goods and can be created without the need for a central authority. As such, Open Source has become a part of the political sphere. Political systems should be designed so that they serve society even if the stakeholders involved—citizens, politicians, businesses, and others—act in their own self-interest. Open Source provides the tools and processes to make this possible for the creation of software technology and information goods in general. It fills an economic gap by enabling the decentralised provision of non-market goods potentially at a global scale. By connecting individuals and organisations in the production process, Open Source bridges the formal and informal economy. This enables grassroots and volunteer initiatives to have an impact, as illustrated by the successes of the early volunteer-driven communities. The mechanisms described in this chapter—types of goods, competition versus collaboration, differentiation—are agnostic to economic systems and observable in all of today's societies. There is however a direct relationship between software freedom and individual freedom—self-identification, the way Open Source contributors choose what to contribute and where—depends on civil liberties.[50]

Open Source provisions public goods in the absence of centralised authority. There is a long way to go to realise this ideal, but the first steps are complete. A good next step will be to stop asking 'how can I make money with Open Source' and start asking 'How can I, my business or society realise the benefits from Open Source?'

[50] Karl Raimund Popper, *The Open Society and Its Enemies* (London: Routledge & Kegan Paul, 1957).

16
Business and Revenue Models and Commercial Agreements

Amanda Brock

16.1 Introduction

To understand how revenue is generated in businesses basing their products or services on Open Source licensed software it is necessary to understand what Open Source software is. Armed with that understanding one can consider the application of possible business models. Open Source may be many things, beyond its legal categorisation, but Open Source is not of itself a business model nor was it ever intended to be one. Generating revenue and finding a suitable business model

Amanda Brock, *Business and Revenue Models and Commercial Agreements* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Amanda Brock 2022. DOI: 10.1093/oso/9780198862345.003.0016

for businesses based on Open Source licensed software, is the holy grail to many interested in Open Source today.

This chapter sets out:

- What Open Source is and what it is not;
- A consideration of possible revenue and business models;
- Recent business challenges around Open Source; and
- Commercial contracts.

16.2 What is Open Source?

16.2.1 Overview

The Open Source movement evolved as a reaction to restrictions in sharing and collaborative development of code which arose as a consequence of proprietary software licensing, along with developers' desire to share and collaborate on software development. Developers made a very conscious decision to move away from proprietary software licensing to Open Source. As Mark Shuttleworth, Founder and CEO of Canonical, points out in his Open Source Underdogs Podcast,[1] an individual's experience or engagement with Open Source will shape their opinion of what it is. That may be as part of a community, through a corporate collaboration, or perhaps by engaging with a foundation, each interaction being very different. This disparity of experience and understanding helps us understand why the term Open Source may become confused.

In a legal sense, Open Source is defined by ownership and licensing of software. In the early years of software development, copyright did not apply to software. Code could be freely used by all, not just the creator of the code, without the need for a licence. There was no applicable intellectual property (IP) protecting the code from third-party usage and necessitating a licence to use another's code. From the 1960s onward, copyright has been applied to software through legislation (see details of the evolution of copyright law to code in Chapter 3). The effect of this is to restrict a third-party's use of the software to the grants made by the owner of the software as set out in the applicable licence.

Over time, ownership and licensing of software evolved to form the basis of the legal characterisation of software splitting it into either proprietary or Open Source. A single licence will be categorised as either Open Source or proprietary, never both. That is not to say that there could not be other categories of software licensing in the future but today only these two mutually exclusive licence types exist.

[1] <http://bit.ly/37cR5Cc> accessed 17 April 2022. Any quotations from Mark Shuttleworth in this chapter can be found in this podcast and its associated transcript.

To be categorised as Open Source, a software licence should meet the requirements of the Open Source Definition (OSD) stewarded by the Open Source Initiative (OSI) and the simplest way to ensure this is for it to be approved by the OSI (licensing is explored in detail in Chapter 3). Code licensed on an OSI approved licence is classed as Open Source, and code licensed on unapproved licences is proprietary. When software is made available under an OSI approved licence, the software is Open Source software. Others may try to broaden this definition referring to Free Software, etc., but I find this simplest and attempts to broaden this can be misleading or the consequence of intentional Fear, Uncertainty and Doubt (FUD).

Software can however be 'dual-licensed' and distributed on both an Open Source and a proprietary licence and so be characterised in both categories dependent on the licence the code is distributed on, whereas a licence cannot be both.

16.2.2 It may look like Open Source but it's proprietary

Code distributed under a proprietary licence is classed as proprietary software but instances of confusion have arisen where proprietary licensed code has been described as Open Source, whether intentional FUD or through a failure in understanding.

16.2.2.1 Public or shared source

Creative Commons licences are not designed for software and should not be applied to software. They form a good basis for commons licensing of non-code assets such as documents (see Chapter 24), but equally Open Source software licences should not be used for non-code assets.

Creative Commons includes a category of Non-Commercial Licensing where the recipient of a licence may use licensed materials but may not use them for commercial purposes. There is no equivalent to this concept in Open Source and there is occasional confusion around this distinction even amongst experienced members of the legal community.

Licences for software that are classified as Non-Commercial cannot meet the OSD as they breach Definition 6, requiring "no discrimination on field of endeavour". On this basis no Non-Commercial licences approved by the OSI exist. In turn, any licence restricting usage to non-commercial purposes restricts a field of endeavour and so is a proprietary licence.

This situation has caused some businesses issues in recent years. As a consequence, lawyer Heather Meeker drafted both the Commons Clause working around Definition 6 and the Server Side Public Licence (SSPL). These were drafted for clients unhappy with the consequences of Open Source licensing on their business (this is explained in detail in section 16.9 of this Chapter). Despite having chosen to license their code on an Open Source licence, these businesses became uncomfortable over time with the consequences of third-party commercial use of that Open Source licensed software. In reality, they shifted away from Open Source to proprietary licensing by adding the Commons Clause or shifting to SSPL.

Public source or shared source licences, like the SSPL, grant access to source code, as would be expected of Open Source software and may meet some OSD requirements but don't comply with the OSD in full. They are not Open Source licences. The OSI confirmed that SSPL is not Open Source[2] and described attempts to say it was Open Source as 'Fauxpen'.

Adding a clause to an OSI approved licence as the Commons Clause does, also renders an otherwise Open Source licence proprietary. Meeker designed the Commons Clause to be added to the Open Source Apache Licence to restrict usage of the licensed code for commercial or for-profit purposes. Its guidance says it is applied to 'specific projects to satisfy urgent business or legal requirements without resorting to fully "closed sourcing" '. The implication is that there are shades of Open Source licensing. But there are no shades of Open Source. It goes on to say that 'applying the Commons Clause to an Open Source project will mean the source code is available, and meets many of the elements of the Open Source Definition, such as free access to the source code, freedom to modify and to re-distribute, *but not all of them. So, to avoid confusion, it is best not to call Commons Clause software "open source"*'. It is however not *best* not to call Commons Clause software Open Source. Commons Clause software is not Open Source and should not be called Open Source. The name itself implies an association with the 'Commons' and therefore with Open which appears disingenuous and an attempt to wrongly imply it is Open Source. The Commons Clause was vilified by the Open Source communities.

Open Source and commons licences are publicly available standards and reusable; proprietary licences are not as a matter of course named and shared in this way. Applying this commons or open naming practice to proprietary software licences has added to confusion and FUD implying this is Open Source. The business history of the evolution of these is considered later in depth at section 16.5.

Whilst Open Source may be many things to different people, it was neither designed to be a business model nor to accommodate business models or revenue generation. Any revenue generation or business model that uses Open Source software is distinct from the software's being Open Source. To be successful the business model must be complementary.

Open Source has disrupted the very lucrative proprietary licensing, royalty-based, business model. This disruption to the proprietary revenue model was

[2] <https://opensource.org/node/1099> accessed 17 April 2022.

met with an unsurprising backlash. For many years FUD was circulated around Open Source in response to the challenge it posed to the proprietary licensing revenue models. There is still an element of FUD today, from some proprietary software companies but many now embrace Open Source, and from companies that built their businesses on Open Source only to decide at a later stage that even where they are successful, the Open Source nature of the licence means they have not generated enough revenue and they have switched future releases to being proprietary.

A second issue emerged as later generations of software developers adopted Open Source as the norm. This generation is referred to by Open Source writer, former lawyer and businessman, Matt Asay,[3] as the 'GitHub Generation'.[4] Unlike the first-generation of developers who consciously and deliberately shifted away from proprietary coding to Open Source, this later generation has been taught Open Source methodologies as the norm when taught to code but apparently not taught the nuances and consequences of licensing or the principles of Open Source. This lack of understanding that developing code in the open or sharing source is not enough to make code Open Source risks the longevity of their favoured development methodology.

Asay writes: 'The GitHub generation seems determined to take open source to its logical conclusion: releasing most software under no license at all. Personally, I didn't like that. I wanted (and still want) people to care about these issues. But most don't.'[5] In reality, education and skills development could fix this issue. JetStack founder, Matt Barker,[6] called out 'Education, Education, Education', in his first report as OpenUK Entrepreneur in Residence and emphasised the need for an understanding of business models and revenue generation. Without this understanding and respect for Open Source's nuances and with the inevitable FUD and pushback from disrupted markets Open Source might not survive beyond this generation. The risk Asay points out is an unintended return to proprietary software. This is not a natural conclusion to the evolution of Open Source but the risk of a lack of understanding of what it is.

Additionally practical issues have added to the GitHub generation's confusion. GitHub is the most popular public repository for distributed development and did not require a licence for code released. It has gone some way to rectify this by adding the statement:

[3] <https://www.linkedin.com/in/mjasay/> accessed 17 April 2022.

[4] <https://thenewstack.io/the-new-stack-what-Open Source-means-for-the-github-generation/> accessed 17 April 2022.

[5] <https://www.infoworld.com/article/3640617/elastic-keeps-ticking.html> accessed 17 April 2022.

[6] Open.UK Founders Forum, 'Entrepreneur in Residence and Founders Forum Initial Findings, November 2021' <http://bit.ly/3FVRDy1> accessed 17 April 2022.

> You're under no obligation to choose a license. However, without a license, the default copyright laws apply, meaning that you retain all rights to your source code and no one may reproduce, distribute, or create derivative works from your work. If you're creating an Open Source project, we strongly encourage you to include an open source license. The Open Source Guide provides additional guidance on choosing the correct license for your project.[7]

This represents a practical step forward designed to support good practice.

16.2.2.3 Ethical licensing and other positive actions with negative consequences

Attempts to modify or extend the OSD have not only been based on commercial challenges. A second example comes from ethical licensing where restrictions apply to usage of licensed code for purposes deemed unethical by the licensor. Irrespective of the rights and wrongs or ethics of the field of usage that is purported to be restricted, code licensed in this restricted way cannot be Open Source. Even if a restriction is well intended, it doesn't comply with the OSD. Ethical licences proposed to the OSI, have not been approved. They should not be considered Open Source but proprietary.

This has most recently been seen with attempts to restrict access to code by those in Russia, following the war in Ukraine and known as 'Protestware'.

Adding a non-binding provision to a licence indicating a desire that the code not be used in a particular sector and embedding the message in the code header alongside the attribution provision could allow the request to travel with the code making this wish known without impacting the OSI approved licence. The downside is that the notice is not binding on a user but a mere request. Bram Moolenaar[8] goes a step further in the VIM text editor using 'Charitywear' where his message to 'Help the poor children in Uganda' is in the actual code itself and visible as the code runs. Like the header notice, this request does not change the licence or stop code distributed under it being Open Source.

A non-binding clause in the licence might achieve the same but risks undermining the licence as a non-approved addition to an approved licence. In light of the Commons Clause, discussion this would be unwise.

16.2.2.3 Public domain

In some jurisdictions there is a concept of public domain where code placed in this is freely usable without restriction. This legal concept does not exist universally and anything not clearly stated in writing runs the risk of confusion meaning

[7] <https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository> accessed 17 April 2022.

[8] <https://en.wikipedia.org/wiki/Bram_Moolenaar> accessed 17 April 2022.

its use will be prohibited in many businesses' Open Source policies. It is always advisable to use a written licence for code distribution and not to rely on this concept.

16.2.2.4 Standards FRAND and Open Source

Awareness of a conflict between closed standards and Open Source is relatively recent (see Chapter 11). As Open Source has become prevalent in Telcos, Mobile Network Operators (MNOs) and their ecosystems including certain Software on Chip companies, the concept of fair, reasonable, and non-discriminatory (FRAND) source patent licensing in Standards has become relevant to Open Source. MNOs are heavy users of standards and the associated SEPs licensed on a FRAND basis, with significant royalty revenue streams attached to these. Patent holders appear to have influenced the development of SEPs with their patents becoming royalty bearing SEPs. As a consequence the forensics of Standards development have become very relevant.

Definition 7 of the OSD requires that '[t]he rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional licence by those parties'. As a consequence of this, it is likely not possible to require a further licence, for example a SEP, being executed to enable the use of the Open Source licensed code.

The MNO network is relatively immature, and over the four decades of their existence they have experienced a constant flux and diminution of their revenue models, from the rise of (Over the Tops) OTTs like What's App to restrictions on roaming charges. Understandably, they are resistant to losing SEP royalties through adoption of Open Source and are fighting back by spreading FUD. Unfortunately, their choice is to retain these royalties, to use Open Source or to find a carve out for Open Source like the Open Invention Network. This structure was used by the US Department of Justice in the Rock Star Consortium acquisition of Nortel's patents.

Currently, unhelpful conversations in standards organisations dominated by MNOs are suggesting the redefinition of Open Source to facilitate their patent royalties. This is unworkable as Open Source is relied on by millions of users. It could of course lead to a new FRAND Source proprietary model. This would be proprietary and not Open Source licensing. Again, Open Source should not be bent or redefined to suit a market even one as wealthy and powerful as the MNOs. The MNOs cannot have their Open Source cake and eat patent royalties.

16.2.2.5 Inner source

This term appeared in the 1980s, and Klass-Jan Stol and Danese Cooper's book[9] on the topic explains converting internal development practices to something

[9] Klaas-Jan Stol and Danese Cooper, *Adopting InnerSource* (Sebastopol: O'Reilly Media, 2018) <http://bit.ly/2rF3H6f> accessed 17 April 2022.

collaborative that uses Open Source behaviours, methodologies, and processes to manage code development but which may create code that is neither publicly shared nor Open Source. It's a great way for organisations to work and many adopt inner source practices as a stepping-stone in their journey to Open Source.

16.3 Business Models and Open Source

16.3.1 Open Source is not a business model

Mark Shuttleworth explains that Open Source cannot be understood by looking through the 'lens of the past' with the perspective of analogue tech, which was proprietary and held in the hands of a few large companies. His 'forward-looking' digital lens sees today's Open Source software in the hands of the many.

Some split Open Source into Community and Commercial Open Source. Community is built by volunteers in free time and Commercial is developed through business to generate revenue. Ultimately creating either has a cost that needs to be met whether by donation or payment without even considering profit. A key challenge for Open Source is covering these costs and generating revenue where the software is licensed RF or given away i.e. how to generate revenue if the secret sauce is not a secret.

Whilst a community may simply need to pay for its own excellence to ensure ongoing maintenance of the code, it will have overheads like community events, legal and governance advice, and, in some cases, paying those who maintain the code. Businesses aim to generate levels of revenue that lead to profit despite having given away the software they develop, and, as Shuttleworth points out, 'enabling your own competition with the benefit of your inventions'. He goes on to say that this can be 'financially and emotionally very draining'. That will be particularly so if the business model is not right for the commercial environment. He adds that 'in the future, if you are not Open Source, you will be niche', and that 'Categories' will be defined in Open Source. Ultimately this requires appropriate revenue and business models to sit alongside the distribution of Open Source.

Open Source as a concept should not be confused with a business model, and assumptions that Open Source ought to deliver a business model fail. In October 2008, Matt Aslett produced the first report on business models and Open Source at 451.[10] Whilst there have been huge shifts in technology and this evolution has had a massive impact on the adoption of Open Source, the associated business models have largely remained consistent.

[10] <https://451research.com/analyst-team/analyst/Matt+Aslett> accessed 17 April 2022.

Wikipedia describes a business model as the way 'an organisation creates, delivers, and captures value, in economic, social, cultural or other contexts'. Open Source is a form of software licensing, a socio-political movement, a way to collaborate, or a development methodology, depending on the lens you approach it with but is not of itself a business model.

I developed a simple '*What if Theory*' to use when discussing setting up a business or open sourcing code that will be used in a business.

What if you build your business on software and share it under an Open Source licence and somebody else uses it?

What if you build your business on software and share it under an Open Source licence and somebody else uses it to make money?

What if you build your business on software and share it under an Open Source licence and somebody else uses it to make more money than you make or even a lot of money?

If the answers to the '*What ifs*' is not that they are OK and fundamental to how Open Source works, or if you do not have a business model that you believe will override any discomfort they cause you, then I suggest you do not open source your business' software and that Open Source is not for you. This will avoid the later pain felt by the likes of Redis Labs or Elastic, explored in section 16.5 of this chapter.

Open Source creates values beyond economic value. These societal values, such as diversity of innovation, community, sustainability (as in the United Nations' Seventeen Sustainable Development Goals), collaboration, and the ability to generate ubiquity rapidly as well as to scale at an unprecedented pace, thereby creating business opportunity, that may however outweigh any downsides.

16.3.2 Open Source communities, forking, and maintenance

Anyone may start an Open Source project, but it would be lonely to do so on your own. A key benefit of Open Source is building a diverse community of contributors to innovate. Not just developers, but creators of documents, community organisers, and governance experts are all needed. Projects were historically frequently self-funded and not set up with the intention of generating revenue. Open Source was effectively the forerunner to crowd sourcing and a basis for the establishment of collective equity models.

As projects grow, they need governance and potentially a fiduciary such as a foundation or a commercial entity (see Chapter 18) facilitating activities like signing binding documents, opening bank accounts, practical organisational activities, as well as protecting IP for that community (see Chapter 9). Some projects morphed into businesses with a commercial sponsor and contributors shifted to being employees. Hashicorp is a great example of this (see section 16.5 later in this chapter).

Communities do not necessarily make life easy for their commercial sponsors. As Sid Sijbrandij of Gitlab points out: 'Open Source has always required a social contract between the owners of the project and the community that uses and contributes to it.'[11] Communities are quick to call commercial organisations to account if they fail to meet their end of that contract.

Communities have a superpower - the ability to fork an Open Source project: taking project code and creating their own version of it under a new name to avoid trademark issues. This is sometimes referred to as 'Lifting and Shifting'. Code, as opposed to money, talks. The version of the code with the most contributions following a fork is generally considered the winner. The ability to keep leadership in check through the risk of the threat of a fork is one of the great strengths of an Open Source community.

Communities today include volunteer and corporate communities and in some cases a mix of the two. The volunteer community, like the KDE or Free Software Foundation Europe communities are discussed by Mirko Böhm.[12] He explores their governance, development, and the potential risk that the corporate sponsors may not align with the communities.

Frank Karlitscheck,[13] founder of Nextcloud, built OwnCloud, raised almost US$10 million of investment, and then in 2016 forked the software he had founded at OwnCloud and walked away from its investor money to set up his new business Nextcloud.[14] Nextcloud operates on a business of around forty employees and a community of over 2,000. Community is everything for Nextcloud. As Frank says, this is very unusual in business. Nextcloud has gone from strength to strength through the pandemic and is seeing significant growth.

Another example of a fork is Elasticsearch which was effectively forked by its commercial sponsor Elastic moving to a proprietary licence in 2021, leaving its Open Source version to be picked up by a community. In contrast to Nextcloud, Elastic's non-employed community was small.

Corporate communities may be evolved by a single company/vendor or formed by collaboration across companies working in co-opetition often utilising a neutral foundation as the home for the code and its governance. Business-based communities have evolved over the last decade or so, causing foundations like Linux and the Eclipse Foundations to transition into stable homes for a range of projects. With Kubernetes to Hyperledger, and code valued at over $16 billion, the Linux in the Linux Foundation has become a misnomer; it is a Foundation of Foundations.

Foundations discern which projects to include as either the code is critical to the infrastructure and eco-system and included due to its utility and purpose, or it has

[11] <https://about.gitlab.com/blog/2019/07/05/thoughts-on-open-source/> accessed 17 April 2022.

[12] <https://jolts.world/index.php/jolts/article/view/131> accessed 17 April 2022.

[13] <https://en.wikipedia.org/wiki/Frank_Karlitschek> accessed 17 April 2022.

[14] <https://zd.net/2SItEwy> accessed 17 April 2022.

corporate backers who will fund its administration and evolution. Corporate participants fund their desired projects through membership, paying to participate at different levels in projects and through employee developer hours contributed to projects.

Code built in the open and code that has been open sourced are different. Some organisations open source their code once created, 'throwing it over the wall'. Other organisations create their code in the open and allow third parties to contribute from the get-go, being truly Open Source in their methodologies as opposed to open sourcing code. Either approach may evolve into a healthy community over time.

It is always worth knowing if there is a community or if code is largely created by a single company. Drupal, for example, has 114,702 users actively contributing to the project, while around 99 per cent of MongoDB's code is written by the company's employees, reflecting their very different natures. The research and development (R&D) benefits of community contributions ought to be critical in any Open Source. A project like Kubernetes, open sourced by Google, is a great example of a community success story and a massive R&D contribution being made from a truly diverse community with global impact, despite having started life as a Google-only project. Considering companies creating Open Source, a lack of community or a community made up of company employees should raise alarm bells. The risk of the company shifting away from Open Source appears greater where this is the case.

Sustainability—(the potential longevity of a well-maintained project)—and risk, where this is a small community or a community that diminishes over time, was thrust to the public eye with the Heartbleed virus and again in 2021 with Log4J (see Chapter 14 for detail). The issue of funding code maintenance is current and important and very relevant to ensuring secure Open Source. In 2016, Nadia Eghbal, published 'Roads and Bridges: The Unseen Labour Behind Our Digital Infrastructure'.[15] Her 'Request for Commits' podcast is worth listening to; if you only have time for one episode, listen to the Finale.[16] In particular, her work focuses on the issue that 'support doesn't scale'. The problem, as Eghbal identifies, isn't just money but is also a question of governance and the need for many hands in a community. Tidelift's Maintainer surveys may offer a more complete and up-to-date picture, the 2021 survey being the most recent at the time of writing.[17] The discussion around the nature of Open Source and security which inevitably requires code maintenance is very alive as the book goes to press.

Tidelift pays maintainers through subscription. Its founders have all worked at Red Hat and their model shows more than a nod to the Red Hat subscription model, with professional maintainers working on Tidelift's code releases allowing business use of Open Source through a fixed-cost subscription to a Tidelift-sanitised and

[15] Nadia Eghbal. *Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure* (New York, NY: Ford Foundation, 14 July 2016).

[16] <http://bit.ly/2tgmcOJ> accessed 17 April 2022.

[17] <https://tidelift.com/subscription/the-tidelift-maintainer-survey> accessed 17 April 2022.

maintained version of the selected Open Source code. Subscriptions are then spent by Tidelift on enhancing and maintaining the packages that they distribute. If there is not a maintainer for an identified package then funds are set aside for a maintainer to join them with that funded incentive. This is subscription for a service as opposed to a royalty for code, with Tidelift acting as an intermediating agent between the maintainers and the subscription payers.

Open Teams,[18] on the other hand, facilitates a marketplace as broker matching service providers with those needing services. GitCoin[19] rewards contribution with quadratic funding,[20] being a form of matching funding for public good, collecting monies from the community, and redistributing them to the community. All are attempts to support the maintenance of Open Source and to support secure and sustainably maintained Open Source.

In small projects, trying to maintain them may be a huge burden on an individual and risk individual burn out or change in circumstances. Tooling automation and allowing automated contributions of course helps this. Tools like Semantic Release create complete release automation. We have also seen a shift in this and improvements on maintenance of projects as can be seen in the Tidelift Report.

With mass adoption in the public sector it may be time to consider the need for government funding at scale, classification of Open Source as a 'public good' as well as being in the 'Commons', and to look to state funding not only to build infrastructure but to ensure adequate long-term maintenance and security of community Open Source projects with mass adoption. The concept of 'Curation' as a generic term for the work necessary to ensure Open Source in public infrastructure is safe and maintained and 'Stewardship' for the holding of this code, were embryonic as the book went to press.[21]

16.3.3 Evolving from a community to a business

Armon Dadgar, co-founder of Hashicorp, describes the evolution of a community into a business beautifully in his a16z.com podcast:[22]

> it didn't start necessarily as thinking about turning the open source into the business. It was more about recognising that there's a clear market gap in terms of, in our case, DevOps tooling … And then realising it's very hard to become a

[18] <http://www.openteams.com> accessed 17 April 2022.

[19] <https://gitcoin.co/> accessed 17 April 2022.

[20] <https://hidorahacks.medium.com/what-is-quadratic-voting-funding-how-did-we-improve-it-70989e813cf9#:~:text=With%20a%20quadratic%20funding%20algorithm,by%20Gitcoin%20quadratic%20funding%20grants> accessed 17 July 2022.

[21] Kate Rawson's Doughnut Economics Action Lab considers this: <https://doughnuteconomics.org/about-doughnut-economics> accessed 17 April 2022.

[22] A16z.com/.

> large sustainable project if you have negative cash flow … if you're solving a large enough problem, you eventually need teams of dozens, hundreds, thousands to work on that problem. You need a business. There has to be a top line connected to your bottom line.

The number of projects starting in this way today, without considering the potential of their code to become a business asset for the founders, is unsurprisingly much less than was historically the case. Clearly it is essential to understand the impact and associated risk from Open Source licensing to a proposed business model and to select an appropriate revenue or business model *before* you choose to open source your software.

Bear in mind that if you open source code, the inevitable consequence is that others including your competitors may freely adopt and exploit the innovations licensed without benefit to your company. You open source with the knowledge that future funders may have different views on revenue generation. Open Source may risk being abused as a marketing tool, to benefit from its facilitating rapid scale and ubiquity, but its communities do not view their contributions as marketing assets.

Businesses' ability to use Open Source code without payment to the provider is fundamental to the mass adoption and success of Open Source in business over the last decade but it creates the risk that a third party may commercialise your code and that 'one day you have a company the next day, that's a feature of a cloud platform'.[23]

Adam Jacobs, explains:[24]

> Let me be 100% clear: this is not a failure of Open Source. This is the deepest, most fundamental truth about Open Source and Free Software in action. That you, as a user, have rights. That those rights are not contingent on the ability of someone else to capture value. That those rights extend to everyone, [including AWS]— or they don't exist at all.

Mark Shuttleworth points out that companies distributing Open Source software need users and adoption or they won't have a business, but this creates reliance on users remembering to 'treat others as you would like to be treated'.

James Watters, Senior Vice-President of Product at Pivotal,[25] suggests that 'if you have the right relationships with customers they will be just as happy to give an Open Source company money as a proprietary one'. Of course this isn't through outdated licensing royalties but services, support, or possibly a subscription model. Open Source allows innovative, high-end, feature-rich, and enterprise-ready software products with unimaginable breadth.

[23] <https://opensourceunderdogs.com/episode-40-pivotal-james-watters/> accessed 17 April 2022.

[24] <https://medium.com/sustainable-free-and-Open Source-communities/free-software-is-the-only-winner-in-elastic-nv-vs-aws-9416f2a0a7f5> accessed 17 April 2022.

[25] <http://bit.ly/39nRpQr> accessed 17 April 2022.

16.3.4 Open Source and businesses

Matt Aslett's 2008 report[26] stated: 'Open Source is not a business model. It is a development and distribution model that is enabled by a licensing tactic.'

The 2010 follow-up added that '[t]here is an increased focus on Open Source as a development model for the creation of software to be monetised indirectly, rather than a licensing strategy to spread adoption for direct monetisation'. 'How is it possible to generate revenue from something that is free?' 'What products and services do Open Source vendors provide that customers are prepared to pay for?'

The Report recognised the following business models with 'vendors' selling services around Open Source, not the code or licences to the code:

- Commercial licences—dual-licensing;
- Subscriptions—annual, repeatable support and service agreements;
- Service/Support—ad hoc support;
- Embedded hardware—software distributed embedded by hardware vendor;
- Embedded software—Open Source is embedded within commercial software;
- Software as a Service (SaaS);
- Advertising—funded by associated advertising;
- Custom development—pay for the software to be customised; and
- Other products and services.

Another of the classic works on the commercial models of Open Source is John Koenig's seven optimisation strategies[27] which includes similar models:

- Dual licensing;
- Support;
- Consultancy;
- Patronage;
- Hosted;
- Optimisation; and
- Embedded.

All have merit and relevance in any assessment of business models today. Since these publications by Aslett and Koenig, and Eghbal's 2016 'Roads and Bridges', there have been a couple of major shifts that hugely impacted business and public sector adoption of Open Source and inevitably impacted associated business models.

[26] <https://451research.com/analyst-team/analyst/Matt+Aslett> accessed 17 April 2022.

[27] John Koenig, 'Seven Open Source business strategies for competitive advantage' (2022) <http://bit.ly/2ZLTPUZ> accessed 17 April 2022.

The first is the pervasiveness of cloud computing creating a level of friction between platform companies and businesses based on Open Source, in particular the Open Core companies.

The second and possibly single biggest facilitator of Open Source adoption in business is the creation of Git by Linus Torvalds and Junio Hamano in 2005. This led to the development and irreversible growth of web-based repositories facilitating dispersed working, for example GitHub and GitLab. They hugely impact business usage of Open Source by enabling engineers to take and use Open Source from these repositories in their organisations without a traditional, often lengthy and outdated, procurement process, the pain of reaching legal agreement, or obtaining financial approval before trying the code. Proprietary code simply cannot compete with this opportunity. The rise of Dev Ops and Git Ops have also had their impact.

Removing friction and the need for understanding risk and benefits of Open Source in procurement has shifted the success of Open Source massively and facilitated its ubiquity. Organisational shift from contracting rules to robust policies and procedures, implementing standards like Open Chain in Licensing (see Chapter 6) and SPDX for supply chain management (see Chapter 7) follows naturally.

For companies distributing Open Source, this has shifted the consideration from how to market code to businesses for their adoption to developing services that businesses will clearly require following adoption with a consequential focus shift from outbound marketing promoting the existence of a product to inbound marketing responding to requests for help and support with the product already being used.

The seminal *The Cathedral and The Bazaar*,[28] Eric Raymond's book on Open Source, emphasised the advantages of distributed, collaborative development compared to the controlled and closed development practices of proprietary software vendors as one of the first texts on Open Source. By 2008, Aslett[29] noted that this Bazaar model was not occurring as Raymond might have expected and the closed Cathedral model remained popular. The shift to the Bazaar in the last decade can largely be accounted for by this practical facilitation through Git and public repositories.

16.4 Commercial or Business Models

16.4.1 Data and its impact on business models

Data is king or the new oil.

[28] <https://en.wikipedia.org/wiki/The_Cathedral_and_the_Bazaar> accessed 17 April 2022.

[29] <https://451research.com/analyst-team/analyst/Matt+Aslett> accessed 17 April 2022.

As Sam Tuke, former CEO of PhPLists, says:[30]

> We've seen . . . companies for which data is the real product, which create services around the data, which empowers them to provide services to consumers for free. They've done phenomenally well, they've scaled incredibly. Open Source products are in quite a good position to compete on that level, because when you have a good Open Source product; you have a wide-scale adoption; it's hard to compete with free, we still have big advantages in terms of distribution, in terms of buy-in from technical decision makers.

Open Source removes the need for paywalls to develop this further, promoting ethical data collection, allowing a data-based asset to be developed, and value to be created. An Open Source product has the opportunity to become the *de facto* standard through mass adoption. If data capture is associated with the services (which in Open Source is not always the norm), then value in that data may offer a clear path to revenue.

The recent focus on data sovereignty, changes in privacy laws, and international data transfers along with geopolitical shifts, mean that the ubiquity of Open Source infrastructure supporting the management of data in a transparent and trusted way facilitates that localisation and control. Federated data models, like Gaia X in Europe, being built on Open Source will enhance this and we are likely to see an evolution of revenue and business models in this area.

16.4.2 Pureplay Open Source businesses

16.4.2.1 Open Source software with services and consultancy

Nextcloud, HortonWorks, and Canonical have been great examples of this, offering Open Source software with support offerings and consulting services as their commercial model, not generally relying on a core with proprietary code or subscription.

'Support doesn't scale', they say, yet founders like Shannon Williams of Rancher[31] made a pure support model work. Rancher's support was part of a lock-change and not sold purely for Rancher's own code but also provided to support alternative code offerings. This pattern is increasingly evident in companies like Percona, which run successful pureplay Open Source-based businesses and multi-product support (i.e. support not only their own but third party software). This multi party model is likely to be the future of support. Businesses like Aiven are betting on it.

[30] <https://opensourceunderdogs.com/episode-23-phplist-open-source-email-marketing-with-sam-tuke/> accessed 17 April 2022.

[31] OpenSource Underdogs podcast (10 December 2019) <http://bit.ly/37stvSp> accessed 17 April 2022.

Their own product is the route to market and being Open Source enables its pace of adoption and scaling up, but Shannon is clear that developing a business around this needs confidence in your product, to walk away from unviable deals, and to pivot the price point to allow the business growth covering the cost of the ongoing development of the Open Source code. He says making the support model work requires two things:

1. useful tech; and
2. a quality product.

Notably Rancher had a 2,000-strong Slack community and interestingly, a bigger user community than a contributor one. Rancher's business grew to 200 staff before being acquired by SUSE in 2020. Companies like Canonical and Rancher have reached real scale in support but they also provide consulting services and feature driven investment from customers.

16.4.2.2 The 'copyleft' model

The 'copyleft' tradition began with the GNU General Public Licence (GPL) aiming to prevent proprietary forks that might be possible with permissive licensing by requiring source code publication on distribution. The Open Source Affero General Public Licence (AGPL) and the related but proprietary SSPL follow copyleft. They are intended to address competitive hosted services and require code publication where there is no distribution and the code is used in a hosted or platform model unlike most copyleft licensing which requires publication on distribution.

Nextcloud's founder Frank Karlitsheck advocates for copyleft business models[32] because they:

- 'Create the best communities. All contributors are equal.
- Mean Forks are always possible, nobody is special. This enforces the best results. Anyone can fork it and make it better.
- Enable a global upstream/downstream network. It allows us to stand on the shoulders of giants. There are no code islands.
- Work with the best developers, because they like Open Source
- Mean Ecosystems are key, GPL creates an equal playing field.
- Create good community governance.
- Remove vendor lock-in. Fair relations lead to happy customers.
- Mean owning the code is not important. The value of a company is in the people.'

Karlitsheck's view is one based on collective equity, and perhaps its transparency allows for the most sustainable business model but perhaps they are not so appealing

[32] <https://www.youtube.com/watch?v=Y3b74UZX5s0> accessed 17 April 2022.

to venture capitalists investing in companies like Elastic and MongoDB. Their main driver may appear to be shareholder value and generating a short-term return on investment. We are living in a time of shift in business and these broader values of Open Source may be more recognised and form the basis of a new economy and capitalism by the time the third edition of this book is published. Certainly, they align with a more equitable and sustainable approach to business.

16.4.2.3 Open Core or COSS

In 2008, Aslett identified that most companies mixed software under both Open Source and proprietary licences, which is the basis of Open Core, and which remains prevalent as a commercial model across Open Source today.

16.4.2.3.1 COSS—commercial Open Source software

Freemium/Premium requires two versions of a product, one free and one with greater functionality which is charged for. It is frequently used as a model in the provision of music or content services, such as Spotify. It is a common business model in start-ups allowing them to hook a customer base with a free product and encourage subscription purchase for add-on services.

'Open Core', commercial Open Source software or 'COSS' is a term coined by Joseph Jacks, founder of OSS Capital.[33] Open Source is *not* freemium, says Jacks. 'Freemium is permissioned. Open-Source is permissionless.'[34] The sentiment of Open Source being unbounded where anyone can take the Open Source code without a further permission beyond its Open Source licence. Open Core is 'limited by feature but not by users' as in Jacks' unbounded Open Source, with a commercial version or add-ons available as proprietary or bundled with services such as support.

To be commercially usable, Open Source software does not need to be Open Core. By definition (the OSD) all Open Source software can be used commercially. Since 2018, certain Open Core companies' reactions to the utilisation of their Open Source code by big cloud companies and, in particular, AWS, have been high profile. Understanding the recent history of the Open Core companies and their actions since 2018 is important and explored at section 16.5 later in this chapter.

16.4.2.3.2 Tight and Loose Open Core

Adam Jacob, CEO of System Initiative and Co-Founder of Chef, describes Open Core as falling into two categories: Loose and Tight.[35] In his 'Medium' post, he contextualises this:

[33] <https://oss.capital/> accessed 17 April 2022.

[34] <https://coss.media/for-an-open future/> accessed 17 April 2022.

[35] <https://medium.com/sustainable-free-and-open-source-communities/free-software-is-the-only-winner-in-elastic-nv-vs-aws-9416f2a0a7f5> accessed 17 April 2022.

'Open Source covers those core values with a sheen of business value. It says that the collaboration and community fostered by those values is a better way of building software, and a better way of building business value' and 'very frequently has little to no acquisition cost—it is both free to receive and trivial to acquire. Open Source says this is a huge business value upside—that getting people easy, low friction access to your software creates a much larger pool of potential customers.'

Riding atop both of these concepts is that of being a community. When we work together on a piece of software for a common purpose, we form a community. In the case where our communities include commercial ambition, we move beyond just thinking about the software's best interest. We also care about our own share of the pie: how much of the money is my money? How much is yours? Is it 'fairly' distributed? If we find a way to work together, for the common good of everyone, we stay together. When we can't, like any other community, we splinter.

16.4.2.3.3 Tight Open Core

Jacob goes on to explain that Tight Open Core allows for the business benefits and value of Open Source including collaboration in the commons, low-friction acquisition of users and growth of a community, with its primary functionality being under an Open Source licence. Direct, often critical, features are only available via proprietary licensing, and fundamentally the code serves the function of the commercial 'master' (note not a commercial sponsor) as opposed to a community. Community exists to fuel the company's generation of economic value as opposed to fulfilling the values of Open Source. In this scenario, the company's interests sit above any community desires or interests impacting the value and sustainability of its community. Unsurprisingly, OSD Definition 6, requiring no discrimination on field of endeavour or use, thereby prohibiting restriction on a commercial entity's ability to take and reuse Open Source, creates a potential threat to their businesses and revenue models.

This also creates a potential risk to other users as 'Tight Open Core is antithetical to the creation of sustainable Open Source communities'.

This is also emphasised by VM (Vicky) Brasseur, who says in her blog:[36] 'If these companies actually cared about the *projects*, they would have invested the resources to build stronger communities around them.' The obvious conclusion is that the stronger a community around a company, the more sustainable its business is.

[36] <https://anonymoushash.vmbrasseur.com/2019/06/07/the-problem-with-amazon-and-Open Source-isnt-amazon/> accessed 17 April 2022.

16.4.2.3.4 Loose Open Core

Jacob suggests[37] that in Loose Open Core, 'you build stuff around the core [Open Source] software, and then monetize that . . . But it was disconnected from the core value prop[osition]; you could still just use the core of the software to do whatever you wanted, without having to do a lot of work.'

In both Jacob's Loose and Tight models, companies and their communities build features around an Open Core, but in the Loose model the Open Source core can be used without the purchase of the proprietary add-on and the core is likely to have more utility on a stand-alone basis. Here the friction between community and corporate interests is less and the company aligns more with the values of Open Source. This Loose Open Core kind of model has more likelihood of being sustainable as Open Source.

The value of community versus pure economic gain is at the heart of this.

'I think being Open Source is not enough to create that community on its own. But that community can only exist around products that are Open Source. Because otherwise you're a fan. And I'm a fan of so many things, but they're not my church; they're not my place, they're not my people', Jacob writes.[38]

As Karlitsheck recognises, the non-economic or societal values, such as community, are fundamental to pure-play Open Source. It's increasingly clear that companies using Open Source in business without subscribing to the values (including Community) build their Open Source on their own, with limited contributions and visibly small communities.

These companies pay lip service to Open Source and are those most likely to use (perhaps abuse) Open Source simply to gain traction and adoption which may lack depth or understanding of the non-economic benefits and be nothing more than a transitory marketing tool.

When Chef moved from Open Core to a pureplay Open Source model, Adam Jacob wrote:

> I couldn't be more thrilled. For me, it eliminates the longest-running source of friction and frustration from my time at Chef. On the one hand, we have a community that cares about the software, and about each other, where we develop the software in concert with our users and customers. On the other, we produced a proprietary software stack, which we use to make money. Deciding what's in, and what's out, or where to focus, was the hardest part of the job at Chef. I'm stoked nobody has to do it anymore. I'm stoked we can have the entire company participating in the Open Source community, rather than burning out a few

[37] <https://changelog.com/podcast/353> accessed 17 April 2022.
[38] <https://changelog.com/podcast/460> accessed 17 April 2022.

dedicated heroes. I'm stoked we no longer have to justify the value of what we do in terms of what we hold back from collaborating with people on.[39]

16.4.2.4 Enterprise open source and the subscription model

Enterprise Open Source software is defined by Joe Brockmeir as an enterprise 'product [that] requires testing, performance tuning, and [to] be proactively examined for security flaws. It needs to have a security team that stands behind it, and processes for responding to new security vulnerabilities and notifying users about security issues and how to remediate them'.[40] In an Enterprise Edition its community may be the business' main competitor, but increasingly third-party competitors offer services and support not only for the code they develop but also for those third-party products.

In a subscription model, whilst the software is technically Open Source, add-on services are provided in this duality. The distro may not be available without paying for a service and the service will be curated to meet professional needs such as legal and governance requirements, security, support, and up-time, etc. This will often be a bundle of services.

Red Hat is the best-known business using this model. It was sold to IBM for $34 billion in June 2019, the biggest tech transaction in history up to that point of time. Its success in subscription with Red Hat Enterprise Linux is one of the biggest economic successes in Open Source.

SUSE, which describes itself as the biggest independent Open Source company IPO'd in 2021. It also follows this model. Tidelift follows a similar model but with a twist, providing a subscription model for a suite of sanitised o curated third-party software packages which it pays maintainers to update, etc.

16.4.2.5 Certification, trademark licensing

Learning platform Moodle is a great example of Open Source with revenue generated from its platform which allows anyone to build a learning management system, whilst as founder Martin Dougiamas says:[41] 'The business model of Moodle is totally designed to support the project but still achieve[s] the mission of providing the software for free, Open Source, and has all the benefits that it does.' Moodle works like a not-for-profit with all funding going back into the organisation, and

[39] <https://medium.com/@adamhjk/goodbye-open core-good-riddance-to-bad-rubbish-ae3355316494> accessed 17 April 2022.

[40] <https://www.redhat.com/en/blog/what-enterprise-open-source> accessed 17 April 2022.

[41] <https://www.edsurge.com/news/2017-05-02-why-moodle-s-mastermind-martin-dougiamas-still-believes-in-edtech-after-two-decades#:~:text=The%20business%20model%20of%20Moodle,the%20benefits%20that%20it%20does.&text=Most%20of%20our%20income%20comes,certified%20companies%20that%20do%20services> accessed 17 April 2022.

'[m]ost of [their] income comes from Moodle partners, which are certified companies that do services.'

In this model certification is based on trademark licensing and the use of a registered mark being licensed in a commercial context (see Chapter 9).

16.4.2.6 Embedded software embedded in devices

Software is embedded in the firmware or chip in a device, in order to control the device which might not generally be considered to be a computer, and in a world increasingly focusing on the Internet of Things (IoT) and smart devices, it is something that is only going to become more common. Whilst some of the shift to Open Source may have been unintentional, today there is an ever-increasing and intentional move to the use of Open Source software in these devices. Key to compliance with licensing and Open Source governance is of course understanding what Open Source is being used and there is an increased focus on supply chain and bill of materials.

Understanding the code used or distributed is important in all utilisation but particularly so here, as code licences, particularly copyleft, come with obligations to make the source available and the need to consider interactions with proprietary software used. Failures in this area have led to litigation (see Chapter 5).

Revenue is generated from device build savings and with good governance in place including maintenance on an ongoing basis, Open Source may generate significant cost savings.

16.4.2.7 Foundation and financial fiduciary models

Foundations (see Chapter 18) have rapidly become the acceptable code hosting vehicle for corporate collaboration in Open Source.[42] Foundations like Linux, Apache, and Eclipse host projects alongside sub-foundations such as Linux Foundations' Cloud Native Code Foundation.

Cost management and governance issues are removed and foundations may act as fiscal sponsors supporting smaller projects. From very US-orientated beginnings there has been a shift to Europe with Eclipse Foundation moving the entire company and operations there, whilst the OASIS Standards Body and Linux Foundation set up entities in Brussels, and China set up the Atom Foundation.[43] In the US this model is accompanied with tax breaks and advantages.

One example of collaborative or crowdsourced financing through a foundation is the GNOME Foundation, which twice raised collective funding to fight litigation, recently raising $150,000 for a patent defence.[44] Not all fundraising is defensive and many Open Source projects have relied on collective contributions. Everyone's contribution in a healthy community is different depending on the

[42] <https://foundation.gnome.org/2020/05/20/patent-case-against-gnome-resolved> accessed 17 April 2022.

[43] <https://www.openatom.org> accessed 17 April 2022.

[44] <https://news.ycombinator.com/item?id=23256240> accessed 17 April 2022.

resources available to them and some contribute skills and time to the collective benefit of Open Source initiatives.

Individual benefactors donating to Open Source projects have also initiated this structure for their projects. WhatsApp founder Brian Acton invested $50 million from the sale of WhatsApp into Open Source Signal creating the Signal Foundation dedicated to helping people have access to private communication through an encrypted messaging app.[45]

16.4.2.8 Software as a Service, open SaaS, platform, and cloud

The Software as a Service (SaaS) model can be somewhat controversial. Vendors host, support, and maintain software in the cloud and charge a recurring fee for hosting and supporting a user's access to the service. A large proportion of the software behind the scenes is Open Source but the services may not be seen to be Open Source. This forms the basis of cloud computing, particularly the public cloud.

Users don't host or access code but receive a service benefiting from it. They may not know what software is used. An open SaaS product has on the face of it the same benefits as a proprietary SaaS product, including cost and potentially environmental savings, rapid deployment, and maintenance and ongoing development by experts whose core business is expertise in the software underlying the service.

Shifting to this model has as an inevitable consequence moving enterprises away from running software on-premise (on-prem).

There is a need for some context and discussions of the impact of this in the last few years.

16.5 Cloud and Open Source in the Last Few Years

16.5.1 Platform companies

Adept use of digital technology isn't enough to make a business successful. In *The Business of Platforms*,[46] a successful platform must have a sustainable business. To be a sustainable business, the authors point out that a platform must be 'financially viable and politically and publicly acceptable'. They must perform better than their competitors. This is also true of all businesses.

The authors split platforms into two types, 'innovation platforms' which facilitate the development of new products and services, by way of building an ecosystem, like Android where monetisation comes from sale of other services; and 'transaction platforms', where the monetisation tends to come from the facilitation

[45] <https://en.wikipedia.org/wiki/Brian_Acton> accessed 17 April 2022.

[46] Michael A Cusmano, Annabelle Gower, and David B Yoffe, *The Business of Platforms* (New York: Harper Collins 2019).

of the sale of goods and services. Of course, some are hybrids, which support both types, that is a combination of product and platform businesses, either in the same business or same platform infrastructure.

Innovation platforms tend to be four to five times bigger and three times more valuable and spend more on R&D as a percentage of revenues, but transaction platforms are growing faster in terms of revenues and market capitalisation and traded at higher ratios of market values relative to sales—investors often consider transaction platforms more valuable relative to their revenue when compared to innovation platforms.

In the case of Open Source, the platforms we are thinking of are generally the former: innovation platforms.

16.5.2 Platform companies and Open Source

The shift from 'on-premise' to 'cloud or platform' technology has massively affected Open Source. The scale and reach of the cloud companies arguably impact a much broader category of business beyond Open Source but the ability through its licensing structure to use Open Source in a platform without contributing back or doing so in a limited way, whilst reaping the benefits, has left many sore.

Cloud servers, artificial intelligence (AI), and IoT are largely built on Open Source. It is the methodology of choice for new innovations. This didn't happen overnight but over decades. Technology previously in the hands of a few companies has evolved into the hands of the many. Platforms have become cost effective partially by sharing the cost of development through Open Source.

Pre-pandemic key sectors had digitalised. Today, almost all businesses have digitalised and many recognise that they, their competitors, and even other sectors need the same digital infrastructure and technologies to allow operations in this new digitalised world. They require the same features. The cost of developing those are driven down through use of Open Source creating cost-effective and non-differentiated platforms through collaboration (see Chapter 15). Cost is the wrong focus, despite it being the most common, and the values of collaboration include skills development, diversity, and building better and more innovative and sustainable code as well as 'community', and the ability to recycle and reuse.

Open Source is the submarine that powers the digital economy by stealth. It is there and everything is built on it, but perhaps the C-suite is not yet fully aware of its true value to their businesses.

Understanding this non-differentiated use of a code stack and the value of Open Source across a sector as the sector digitalises is an understanding that develops gradually, often over a two- to five-year period. In early-stage digitalisation, a sector moves to development of code at scale or as infrastructure, and a business or a sector will not understand what really matters. That understanding develops with time and experience. As participants ultimately drive to the same innovation goals, costs are saved, and better code produced through collaboration on digital development.

Simon Wardley[47] was Director of Cloud at Canonical in 2010 and he often discusses the overwhelming success Canonical had, gaining over 70 per cent of the cloud operating systems market almost overnight. A runaway success.

However, the Ubuntu operating system was 'free' using Open Source licensing with no royalty charged for usage and Canonical's support-type business model had understandably been developed for a different on-premise environment. Cloud was relatively new and evolving. It relied on the need for implementation and other engineering, consultancy, and support services being bought by enterprise users for configuration, implementation, and customisation. These were not needed in the cloud platforms and the success was 'free'.

Support and subscription differ as the software that support is sold for is readily usable without a subscription, whereas a subscription-based product is likely sold as enterprise-ready. Cloud companies could use the enterprise-ready Ubuntu without buying support services. This was a key differentiator from Red Hat's subscription model, requiring regular payment for a useful operating system.

As cloud companies do not generally distribute code to their users but build their services on top of it and sell services (SaaS), the copyleft obligation triggered by distribution of Open Source licensed copyleft code in licences like the GPL is not triggered by this kind of usage.

Canonical founder Mark Shuttleworth[48] had the personal means to fund the Canonical commercial sponsorship of Ubuntu and business. This may have had an unexpected consequence in the cloud story as Canonical's experience may, despite being an early-stage warning to other providers, not have been obvious to them.

From 2010, this shift to the cloud or platform economy grew and grew. Many companies set up businesses and developed software as Open Source then built businesses around it, years after Canonical experienced the impact of this new cloud model.

A decade after detailed discussions of business models for Open Source based businesses had begun considering the impact of cloud and its interaction with Open Source, understanding Open Source licensing and the nuances and responsibility of community contributions, might reasonably be considered to have been a necessary basic consideration in open sourcing code or setting up a business based on it.

In 2019, Matt Asay described Open Source as being what underdogs do to win.[49] In 2021, after a stint at the cloud company AWS, Asay returned to his

[47] <https://en.wikipedia.org/wiki/Simon_Wardley> accessed 17 April 2022.

[48] <https://en.wikipedia.org/wiki/Mark_Shuttleworth> accessed 17 April 2022.

[49] <https://www.techrepublic.com/article/whats-really-behind-microsofts-love-of-Open Source/> accessed 17 April 2022.

former employer, MongoDB (which has a Tight Open Core model). Around the same time,[50] he describes companies contributing to open source as 'selfish', explaining that:

> [d]evelopers may contribute for the sheer love of code; companies don't. Never . . . Resources are finite. If a company spends money and resources to contribute code, it's because they've done the math and believe they'll earn a return on that investment . . . The reason will be as with Google and others, to help drive greater customer adoption of its own products. This is just how (Open Source) business works.

Companies do indeed contribute because it makes good business sense. Others argue that it is not those contributing who are selfish but the companies who use Open Source without contributing back who are selfish, and indeed short sighted.

16.5.3 Open Core and strip mining

16.5.3.1 Background

In the last few years a number of Open Source, mainly Open Core, companies complained that the use of their Open Source software by cloud companies attracted inadequate financial compensation or contribution back to the upstream Open Source project from those cloud companies. The companies complaining were making money but were not happy with how much of the revenue cloud companies generated was shared with them. Some of the most vocal companies, Elastic[51] and MongoDB amongst them, were very profitable before the allegations and continue to be now.

SaaS Squatting[52] describes when a large SaaS or cloud company uses Open Source technology, which is freely available to it but doesn't contribute financially in return despite the Open Source usage generating large revenues, even if this failure to share effectively undermines the company that's developing the Open Source software being used.

Strip mining is a US term. It is an analogy to a federal law from the mid-1970's designed to curb abuses by companies strip mining the land for coal in an abusive way. The alleged 'strip mining' small companies' software and in particular Open Source software by large cloud companies has become a very public debate.

[50] <https://www.infoworld.com/article/3632360/Open Source-is-selfish.html> accessed 17 April 2022.

[51] <https://www.infoworld.com/article/3640617/elastic-keeps-ticking.html> accessed 17 April 2022.

[52] Michael Schwarz of Gluu and host of Open Source Underdogs.

A group of seven founders and CEOs including Elastic and MongoDB's thought this use of this software was an abuse, according to the *New York Times*.[53]

Amazon was equally vociferous in its response that Open Source licences allowed its use of the code in this way and that it had done nothing wrong.

Whilst strip mining coal may be illegal under US federal law, using code whose originator has freely chosen to license it under an Open Source licence is entirely legal as pointed out by AWS's Andy Guttman in AWS's blogpost response.[54]

The seven CEOs apparently considered suing Amazon. One of those, from Elastic, had already commenced litigation for trademark infringement of its Elasticsearch product which settled in February 2022, without details being shared. The story appearing in the *New York Times* is indicative of the importance of Open Source to the cloud infrastructure at this juncture in history, combined with the platform companies' power and need to use Open Source as well as the high sums at stake.

16.5.3.2 MongoDB and SSPL

Database company MongoDB changed its licence from the Open Source AGPL to the proprietary SSPL, which requires the source code of the entire service be released under the SSPL, if it incorporates an SSPL-licensed component. Bruce Perens,[55] co-author of the OSD, argued that the SSPL violated Definition 9 of the OSD, that the 'licence must not restrict other software' as SSPL forces any SaaS software aggregated with the SSPL licensed software, even if not a derivative of that software, to nevertheless be Open Source.

The SSPL was withdrawn from the OSI's licence approval process so was not rejected by the OSI but equally was not approved. It was probably removed as its authors recognised that it would not receive OSI approval. It is therefore a proprietary licence. Several major Linux distributions dropped MongoDB after the change.

John Mark Walker[56] says that the 'emergence of the Open Source business model as a distinct class pushes companies that adopt it into a narrow-banded decision matrix that presents limited options for future changes if the need to pivot arises'. In the emergence of cloud and Open Core, certain companies have given away their IP assets in their entirety via Open Source licences without first having a robust business model (perhaps it would be helpful to consider my 'What If' analysis, at section 16.2).

[53] Daisuke Wakabayashi, 'Prime leverage: how Amazon wields power in the technology world' *New York Times* (16 December 2019) <https://nyti.ms/2u1BXti> accessed 17 April 2022.

[54] <https://aws.amazon.com/blogs/opensource/setting-the-record-straight-aws-open-source/> accessed 17 April 2022.

[55] <https://en.wikipedia.org/wiki/Bruce_Perens> accessed 17 April 2022.

[56] <https://medium.com/@johnmark/OpenSource-business-models-considered-harmful-2e697256b1e3> accessed 17 April 2022.

Once the decision to use Open Source software is made, it is not possible to take back what has been given Open Source. It is possible to shift future distribution but the places to shift to are limited and generally mean to a proprietary place where focus purely on revenue has trumped the benefits of community and collaboration.

Dev Ittycheria, CEO of Mongo DB, was clear that 'We [Mongo] open sourced as a freemium strategy; to drive adoption'. He effectively acknowledges that Open Source was no more than a marketing strategy to MongoDB, a statement he later confirmed he did not regret making despite its causing huge offence to communities whose work had created the MongoDB products. Mongo's stock rose 203 per cent between its announcement that it would be changing the terms of the licence to a proprietary licence in October 2018, and June 2020.[57] That does not mean it would not have seen the same trajectory had it remained on its Open Source licence. We will never know.

In Mongo's case there is little opportunity to give the benefit of the doubt and consider this a lack of understanding. MongoDB's community contributions were very limited and its Open Core model was Tight Open Core.

16.5.3.3 Redis Labs and the Commons Clause

Redis Labs made its Redis plugins subject to the 'Commons Clause', a restriction on the sale of the software which it added to the existing Apache Licence terms. The Commons Clause was created for Redis in 2018 as a response to platform or cloud companies.

Following criticism, this was changed in 2019 to the 'Redis Source Available Licence', a proprietary licence which forbids sale of the software as part of 'a database, a caching engine, a stream processing engine, a search engine, an indexing engine or an ML/DL/AI serving engine'. The last versions of the modules licensed solely under the Apache Licence were forked and are maintained by community members under the GoodFORM project.

16.5.3.4 Cloudera

Cloudera and Hortonworks merged in 2017. Cloudera was a major contributor to Open Source projects, but its business model was based on selling licensed software. Hortonworks, however, sold support and services for Open Source software on a subscription basis. A blog by Charles Zedlewski and Arun Murthy in 2019 and updated in 2020 says:

> Prior to the merger, the two companies distributed their products under somewhat different Open Source licensing models. Aligning the two models was one of the last items on our merger to-do list. Meanwhile over the past few years,

[57] <https://www.protocol.com/enterprise/mongodb-OpenSource-database> accessed 17 April 2022.

> we've seen many of our industry peers revise their Open Source licensing strategies, … plan to consolidate and transition the small number of projects currently licensed by Cloudera under closed source licenses to Open Source licenses

and '[a]ll of our Open Source licenses will adhere to one of two OSI approved licenses: the Apache License, Version 2, or the GNU Affero General Public License, Version 3 ('AGPL'). We considered a modified Open Source license but determined that it was important that we use community-accepted licenses.'

16.5.3.5 Confluent

Some components of the Confluent Platform moved from Apache 2.0 to the Confluent Community Licence (CCL)[58] but this does not impact Apache Kafka, the core Open Source software. This move was less controversial, despite being a move away from Open Source in as much as Confluent did not claim that the CCL is an Open Source licence.

16.5.3.6 Chef

Chef was Open Core and moved to pure Open Source and Apache licensing in 2019, but then moved back to Open Core, removing its free distro and shifting to subscription, along the lines of the Red Hat model.[59]

16.5.3.7 Cockroach Labs

In 2017, Cockroach's website said:

> It's a delicate balancing act. Building paid 'enterprise' features for Open Source software can feel dirty. Paid features diminish the Open Source appeal and can lead to substantial community angst. On the other hand, it's disheartening to see mammoth cloud service providers repackaging OSS for substantial gain without finding ways to foster the Open Source ecosystem, or hundred-billion-dollar multinationals foregoing support licenses from struggling OSS companies. If you're serious about building a company around Open Source software, you must walk a narrow path: introduce paid features too soon, and risk curtailing adoption. Introduce paid features too late, and risk encouraging economic free riders. Stray too far in either direction and your efforts will ultimately continue only as unpaid Open Source contributions.

It goes on to state: 'Enterprise features we introduce will be contained in source files covered by a new license, called the CockroachDB Community License (CCL).

[58] <https://digitizingpolaris.com/is-confluent-still-Open Source-72c579d0d7dd?gi=639c1c0191c0> accessed 17 April 2022.

[59] Adam Jacob, 'The war for the soul of open source' *The Changelog* <https://changelog.com/podcast/353> accessed 17 April 2022.

The source code will still be available, but because it does not include the free redistribution right, it's not Open Source by definition.' Showing a real understanding of Open Source and clarifying that a ' "pure' FLOSS distribution will also be available, with enterprise features absent, for those that require it.'

In June 2019, Cockroach made this public statement on its website:

> our past outlook on the right business model relied on a crucial norm in the OSS world: that companies could build a business around a strong Open Source core product without a much larger technology platform company coming along and offering the same product as a service. That norm no longer holds.

Cockroach Labs then moved from AGPL to MariaDB's proprietary Business Source Licence (BSL), whilst being advised by Michael (Monty) Widenius, Maria DB's founder. On this shift away from Open Source to a very clear Open Core model they commented:

> In order to continue building a strong Open Source core, this restriction has a rolling time limit: three years after each release, the license converts to the standard Apache 2.0 license. Our goal in relicensing with a time restriction is two-pronged: to simultaneously create a competitive database as a service (DBaaS) while also providing a guarantee that the core product will become pure open source.

Interestingly, the BSL licence requires users to license/pay for support in particular circumstances but has no real teeth to monitor or enforce the requirement to obtain a commercial licence and requires the user to act honourably and contact Cockroach to purchase a commercial licence.

As Cockroach founder, Peter Mattis said in his Open Source UnderDogs podcast,[60] they had not yet had to deal with issues with the platform companies but could see these approaching, in a couple of years' time as they scaled. The Cockroach website is very clear:

> Competitors have always been legally allowed to offer another company's OSS product as a service. Now, we're finally seeing it take place. We're witnessing the rise of highly-integrated providers taking advantage of their unique position to offer 'as-a-service' versions of OSS products and offer a superior user experience as a consequence of their integrations. We've most recently seen it happen with Amazon's forked version of Elasticsearch. Another option we considered was adopting a three-tier model: Open Source core, enterprise components, and a middle ground of features that are not open source but available at no cost. This

[60] <http://bit.ly/2sFWkf7> accessed 17 April 2022.

> is a popular model, and more or less the model we have seen until now. However, this creates bad incentives for our company. It creates pressure to avoid creating new features in the core and do as much work as possible in the non-Open Source components. Basically, we would be tempted to sell our core users short by permanently putting our most exciting features behind an enterprise license. If you squint towards the horizon, this model does not bode well for the Open Source ecosystem.

When asked if he would start another Open Source company Peter's response—'yes, but as a SaaS company', not on-site—was interesting and very telling. He pointed out the difficulties in making money out of a pure Open Source model for companies beyond Red Hat with its paid (distro including) subscription, and the possibility that the state of the technology means that SaaS is a constant opportunity at this point, and the only viable one for Open Source.

16.5.3.8 Elastic

The first line of Elastic's 'About' page[61] states: 'Open Source is how a project like Elasticsearch goes from a few downloads in 2010 to over 250 million in 2018', acknowledging the value in business creation and engagement that Open Source brings to any new product. Undoubtedly increased uptake occurs through its simple licensing, zero cost, and process simplicity, all of which 'facilitates exceptionally rapid adoption'.

The merits of Open Source are lauded in terms of product quality. 'But Open Source is not just an effective way to distribute software. It's an effective way to make the best product possible', and community 'building software is good but building a community around your software is better'.

> Building an Open Source business can have its challenges, and we've learned a lot from observing others. Some Open Source companies base their bottom line on a support-only business model. We believe this approach puts what's best for the company and what's best for the user in direct conflict with one another, where one can only succeed if the other struggles. This approach lacks the incentive to make the product easy to use or empower customers to be successful since the company's revenue is based upon customers needing regular support. Some Open Source companies fork a commercial (or 'enterprise') offering from an original open source project. We believe this fractures the project code and community. It also undermines community trust, limits product testing, and dilutes product quality, ultimately competing with the business efficiency gained from

[61] <https://www.elastic.co/about/why-Open Source> accessed 17 April 2022.

> Open Source software in the first place. So, we've built our business differently. Ours aims to strike a healthy balance between Open Source and commercial code in a single, open software stack in addition to support and services. It is up to us to deliver users with enough value (and therefore reason) across all our offerings to invest in us. As a result, we can engineer products that are easy to use and reliable, empower our users to be skilled and knowledgeable, and still be a successful company.

Elastic described this approach as 'dev-first not dev-only', in other words it relied on 'Developer Zero' to upload and kick the tyres of its Open Source software then expect adoption to happen more widely across the organisation until '[e]ventually, a decision-maker or executive takes notice and makes a call on whether or not to formally invest in Elastic'.

Dev-first does not require an Open Core model, and indeed all Open Source based businesses and business models rely on this ability to gain traction. In February 2018, Elastic Founder and CEO, Shay Bannon, announced in a blogpost[62] 'that whilst the company and his commitment to Open Source went deep, Elastic had created a business around the technology which could manage investment and would choose who it charged to use its code and who could use that for free'.

An Enterprise Edition was not the business model of choice at Elastic and they moved to Open Core, producing commercial (proprietary) software that would move to open over time. 'High-value features' would be identified and offered as commercial extensions to the core software, which remained open. The commercial extensions would be licensed on the proprietary 'Elastic Licence Agreement' and identified tools and packages moved to an Apache licence and open sourced over time.

In January 2021, however, Bannon found himself in the eye of a storm when he released a second blog post, 'Doubling Down on Open Part Two',[63] the headline implying a greater commitment to Open Source activity when the company was shifting further away from Open Source and moving Elasticsearch to the proprietary SSPL licence.

> This license change ensures our community and customers have free and open access to use, modify, redistribute, and collaborate on the code. It also protects our continued investment in developing products that we distribute for free and in the open by restricting [Cloud] service providers from offering Elasticsearch and Kibana as a service without contributing back.

[62] <https://www.elastic.co/blog/doubling-down-on-open> accessed 17 April 2022.

[63] <https://www.elastic.co/blog/licensing-change> accessed 17 April 2022.

Bannon goes on to point out that:

> [t]his change in source code licensing has no impact on the overwhelming majority of our user community who use our default distribution for free. It also has no impact on our cloud customers or self-managed software customers. In recent years, the market has evolved, and the community has come to appreciate that Open Source companies need to better protect their software to continue to innovate and make the investments required.

All in all, a very clear and directed action aimed at the cloud platforms.

This resulted in the original Open Source licensed Elasticsearch effectively being forked. The newly SSPL licensed version of Elasticsearch is the fork. The original code has been renamed Open Search and is maintained by AWS, a move described by Salil Deshpande in Tech Crunch as both 'self-interested and rational'.

VM (Vicky) Brasseur explained:[64]

> These projects are not being relicensed to protect them from Amazon. Claiming that they are is at best naive and at worst wilfully lying. These companies are relicensing projects to cover for the fact that they are ignorant of how to run a successful business. They knowingly released their secret sauce under permissive licenses and have discovered that doing so means that competitors can create more compelling product offerings based upon the same technology. This is entirely in accordance not only with the licenses that these companies knowingly chose, but also with a competitive market. The only problem with this is that it came as a surprise to these 'Open Source' companies and now they're reacting poorly.

Use of the Elasticsearch name by Amazon historically had triggered trademark litigation, which Elastic settled in 2022 without details being shared but the existence of which serves as a reminder of the importance of trademark registrations, policy, and strategy.

In 2021, Matt Asay wrote:

> For all the sound and fury, Elastic, the company, seems to be doing quite well. Elasticsearch, the code, doesn't seem to be struggling, either ... sometimes we imagine a developer's choices are strictly binary: open or closed. But as the Elasticsearch example suggests, developers aren't nearly so simpleminded.[65]

[64] <https://anonymoushash.vmbrasseur.com/2019/06/07/the-problem-with-amazon-and-Open Source-isnt-amazon/> accessed 17 April 2022.

[65] <https://www.infoworld.com/article/3640617/elastic-keeps-ticking.html> accessed 17 April 2022.

Asay's implication of a half-way house between Open Source and proprietary code is unhelpful and at the present time misleading. Code is either Open Source or proprietary. Whilst there may be room for a future evolution of a third categorisation, that does not yet exist.

Asay also references a number of companies who continue to use the Open Source version of Elastic's product like Aiven, whose Lorna Mitchell says:

> We've been in with AWS on the OpenSearch fork since the beginning, partners and library maintainers, not actually core maintainers yet, and we are switching all our Elasticsearch customers to our new managed Opensearch, which they seem happy about. We have an upstream contributor in our OSPO, I'm named in the releases. My personal take is that when people buy a service they might not be so bothered—but a lot of people were running Elasticsearch themselves on-prem and it's that quiet majority that will move to OpenSearch, rather than the Elastic Cloud subscribers.

Notably this would not show up in the economics of MongoDB's service proposition.

16.5.3.9 Grafana Labs

Following on from the Elasticsearch controversy, Grafana Labs also reacted to the cloud situation in April 2021. Perhaps with the benefit of hindsight and Bannon and Elastic's experience, Raj Dutta shifted to the Open Source AGPL licence and carefully managed interaction with the community and public announcement on the move, with a Q&A from staff to Dutta shared publicly.[66]

> We believe in Open Source and are not in the business of trying to redefine what that means. AGPLv3 is an OSI-approved license that meets all criteria for Free and Open Source Software. SSPL is not OSI-approved. We considered SSPL and watched community response to the decisions that MongoDB and Elastic made. We really respect their decisions, but we've decided that we want to keep Grafana Labs software under OSI-approved licenses, because the support of the Open Source community is very important to us.

16.5.3.10 Conclusion

There has been FUD around this topic. VM Brassuer eloquently sums it up: 'This is a big steaming pile of bullshit … Fingers need to be wagged here but not at Amazon.'[67]

[66] <https://grafana.com/blog/2021/04/20/qa-with-our-ceo-on-relicensing/> accessed 17 April 2022.

[67] <https://anonymoushash.vmbrasseur.com/2019/06/07/the-problem-with-amazon-and-OpenSource-isnt-amazon/> accessed 17 April 2022.

Open Source was never designed to be a business model and cannot be relied on to provide one. A careful understanding of potential business models and a decision-making process that considers the potentials of these and the risk of sharing code that will potentially be the business' crown jewels ought to be gone through before code is made Open Source or built on an Open Source model.

Giving the last word on this to Adam Jacobs, of Chef, who said:

> Let me be 100% clear: this is not a failure of Open Source. This is the deepest, most fundamental truth about Open Source and Free Software in action. That you, as a user, have rights. That those rights are not contingent on the ability of someone else to capture value. That those rights extend to everyone, including AWS—or they don't exist at all.

Jacob's analysis is very compelling.

The cloud and the platform economy has clearly created issues for Open Source, but they are not the first and won't be the last challenge to its definition and parameters.

A decade ago, operating system wars, bundling and other tactics to close down Open Source challengers in the mobile, desktop, and enterprise markets were top of the agenda. To my mind such challenges will ebb and flow as new technologies and models evolve. Open Source cannot be redefined to meet each challenge that comes along. Inevitably, the adoption of Open Source will shift markets.

16.6 Standards and FRAND

The next challenge is already upon us: standards and the friction of FRAND licensing and Open Source licensing. A whole chapter has been devoted to this.

As a market the MNOs are only a few decades old but are vast and wealthy. They have seen more pivots and changes in revenue generation than almost any market, with changes in roaming tariffs and the challenges of the OTTs like WhatsApp putting services 'over the top' of the mobile networks, and eating their lunch and their data revenue stream. At this juncture, they are unlikely to give up the very lucrative SEP and FRAND royalty revenue without a fight. The additional licence requirement of FRAND does not sit with Open Source, and attempts by the MNOs to have their cake and eat it, by benefitting from Open Source yet retaining these royalties, are doomed in the long term to fail, but not without a fight (see Chapters 3 and 20).

This area will be one of the key areas to watch in the next few years.

16.7 Open Source Business Models—Diversity and Success

As a consequence of the success of Open Source, we now see it everywhere.

One size rarely fits all in anything, but certainly cannot in a digitalised world where all companies today create, distribute, or have their products consumed using software. All companies have become software companies. If all companies are software companies and the best and majority of software is Open Source, then inevitably all companies might be considered to be Open Source companies. There can be no denying that all companies are Open Source software users. In our platform-driven world, many have huge savings in the total cost of enterprise from Open Source software.

There is a huge diversity in the types of enterprises using Open Source to operate their businesses and manage their infrastructure, from bus companies to banks. In the platform and SaaS economy billions, perhaps trillions of dollars, are saved by the platform model, and each and every one of us benefits from this and the platform companies of course benefit from Open Source. Yet, we have no idea of the value generated. We have no idea of the cost to enterprise and each of their consumers without Open Source underlying their businesses.

Mark Shuttleworth recognised that 'There isn't going to be one approach that essentially enables Open Source to be relevant across the full spectrum of technology, because the full spectrum of technology is a very broad spectrum. And so, people yelling at each other and saying, 'You don't understand' is probably mostly a symptom of the fact that people are coming from lots of different backgrounds and making naive assumptions about what it's like in the other person's shoes.

16.8 Measuring Success and the Values of Open Source

16.8.1 Total cost of ownership or total cost of enterprise and other economic descriptions

When we look at the measures of success in businesses around Open Source, we often see the economic term 'total cost of ownership' (TCO). It is even used as late as 2021 in the European Commission Study on the value of Open Source.[68]

The Commission's first working group on Open Source, in 1998, focused on:

> Total Cost of Ownership, because IT platforms were mostly the same for every company and most proprietary software vendors were focusing on TCO alone to push their own IT infrastructure. It was the most widely used economic

[68] <https://openforumeurope.org/open-source-impact-study/> accessed 17 April 2022.

> framework for measuring things. The overall idea behind TCO is simply not adequate today, because the concept of IT infrastructure is not the same anymore. We are not measuring productivity or the value generated by their employees. Companies have a whole different perception of what the user is, Carlo Daffara explained.[69]

Today, many researchers are working on a shift away from this method of economic valuation of Open Source and moving from a GDP-based and inbound (lines of code created and number of developers) approach to investment and utilisation.

There is some discussion of the economics of Open Source in Chapter 15, but like commercial models value calculation is likely to evolve significantly before the next edition of this book. Increasingly research is also looking at the societal values of Open Source too.

16.9 Open Source and Commercial Contracts

The contractual issues and requirements around Open Source software are largely simpler than might be expected. Key to understand is that the contract being drafted is most likely to be a service contract, not a software licence. Open Source software is provided on an OSI approved licence and any services provided by an associated business will be under a service agreement, not a software distribution agreement (unless additional software is being commissioned). Being clear on this avoids the vast majority of inappropriate negotiation around software distribution which one would expect in a proprietary software agreement. Contracts used for agile software services are generally most appropriate and focus on governance and regular updates and decision-making.

More than once during my time negotiating Open Source contracts, I was approached by opposing counsel after the negotiation, confirming that they wished they had listened to my initial advice not to use their standard proprietary software distribution contracts for Open Source deals. Software distribution contracts are of course not generally appropriate for the provision of services.

16.9.1 Service contract not a licence

In looking at the service agreement there is a single important thread irrespective of the applicable commercial model. The Open Source code which the contracts relate to is already distributed on a licence. Any contract involving this code should

[69] <https://openuk.uk/stateofopen-phase-one-report/> accessed 17 April 2022.

not try to create a licence but ought to consider what software is being delivered and provide a bill of materials list. Whilst this practice was normalised as an appendix to the contract this can be difficult with the scale of software packages to which a contract may apply today and with changes over time.

A great deal of work is happening to standardise these lists of software distributed and to manage supply chain. Previously described as Manifest, BOM, Certificate of Originality, amongst many other names we generally refer to this as a Software Bill of Materials (SBOM). The SPDX International Organization for Standardization (ISO) approved standard is the key standard for Open Source SBOMs. This is further explained in Chapter 7.

There may be a need for provisions around mistaken manifests and inaccuracy in the SBOM.

If additional code is to be developed, then there may be a need for a licence provision for that, and if there is, what this contains depends on whether the deliverable will itself be Open Source, when an Open Source licence will apply or proprietary, which will necessitate the drafting of a proprietary licence within the contract and of course the ownership of the code must be considered.

16.9.2 Open Source Definition

As a matter of good practice, Open Source should be defined, and however this is done the OSD must be complied with. I recommend requiring an OSI-approved licence.

16.9.3 Derivative works: GPL or copyleft compilation issue

The long-time fear of commercial enterprises has been the perceived risk of strong copyleft Open Source code being combined with proprietary code that the owner cares about. The perceived risk is that of a derivative work being created with a requirement for the proprietary code to be shared on an Open Source licence. This legal concept is discussed at length in Chapter 2. However, to my mind, the solutions are practical more than legal and ensuring that a derivative work is not created, if the risk of this is in fact an issue, can be managed by an engineering team's technical and not legal practices.

16.9.4 Open Source licence-specific requirements

Complying with the licence terms is the overarching requirement and this may include practical activities like making source available, updating headers in code, and managing attributions. From a contract perspective, a simple clause requiring licence compliance with appropriate remedies is the best route to achieve this. Like many provisions in an agile contract, the details should be in the governance and form part of a company's policies and procedures around use, distribution, and contribution to Open Source software. The Open Chain project tools are one helpful route to creating these good practices. The focus should not be on listing these out in the body of a contract.

16.9.5 Warranties, representations, and indemnities

These terms have specific meanings under English and Scots law. This is not the same across the US and other jurisdictions. In the UK, the consequences of the use of each of these terms and the breaches of these provisions are different and they should be used with some care and understanding an analysis of which goes beyond the remit of this book.

For Open Source software, applying warranties, indemnities and undertakings that had become the norm in a proprietary software contracts is not appropriate. Instead, a contract for Open Source should reflect the terms of the Open Source licences, which specifically exclude many of the traditional warranties such as fit for purpose or satisfactory quality and liability.

Traditional warranties for the delivery of a service which the contract is for, are appropriate and reasonable.

The risk grid created by Andrew Katz, Carlo Piana, Malcolm Bain, and myself as part of my learning process on joining Canonical[70] is freely available and a good place to start learning about the balance of power in such supply and consumption and what is reasonable in Open Source contracts. Katz has updated and maintained multiple versions of the grid to support your considerations.[71]

The reason many of the warranties are excluded and inappropriate is that the Open Source software, unlike proprietary software, is generally provided free of charge and payment makes a difference to the balance of risk and associated liabilities accepted by the provider.

16.9.6 Subscription or insurance

Liability can only realistically be taken by the supplier for the distribution of Open Source software where there is an economic transfer (payment) directly relating to the software. This may take the form of a subscription model or be paid for

[70] <https://www.jolts.world/index.php/jolts/article/view/10/10> accessed 17 April 2022.

[71] <https://www.jolts.world/index.php/jolts/article/view/10>

insurance. The most obvious way in which revenue might attach to the distribution of Open Source software per se is in the sanitisation of the software by a company taking responsibility for it. Companies like Red Hat and Tidelift effectively include a level of liability for some risks in the software within their subscription charges.

It would be extremely unwise for any organisation to take liability for risk beyond the level to which or categories of liability to which it is able to insure unless they are generating revenue of a scale to make self insurance viable. On the other side of the deal, any company relying on either contractual unlimited liability or extremely high levels of liability or indemnity in the event of a risk in Open Source software being triggered, such as an IP dispute, is unlikely to see that reliance being satisfied.

A more appropriate approach is to look for indemnity terms and levels which the provider might reasonably be able to stand behind. If a small company or even a large one takes unlimited or high levels of liability without question, alarm bells should be ringing.

Should liability for a particular risk really be of concern, this is something that should involve insurers and a contractual requirement stating that insurance is bought and renewed for the length of the contract and the ongoing limitation period beyond the end of the contract, during which time the risk would subsist. It may also be wise to note an interest on the insurance policy.

16.9.7 Agile, SOW, and project governance

Again, although somewhat beyond the remit of this chapter it is useful to understand that the traditional approach of a waterfall contract with a rigid Statement of Work (SOW) is not appropriate to the development of projects using Open Source. Strong project governance, regular meeting points, with clear measurables and consequences of these not being met are far more appropriate to a structure for Open Source, which is almost without exception developed in an agile way. Meet frequently, fail fast.

17

Antitrust, Competition, and Open Source

Carlo Piana

17.1 Introduction

Antitrust is a subset of competition law. It mainly deals with failures of the market to work properly because of certain behaviours and actions of the competing firms, said actions being otherwise legal if taken in isolation, but failing the competition test because they have the scope and the effect of limiting competition. Other legislation is directed at ensuring the proper working of the market, including unfair competition, public procurement (where public entities must ensure parity of arms between potential competitors and the best deal in the public interest), public control by independent market regulations in cases where a market can only work under artificial conditions, as in natural monopolies, and state aid.

In this chapter we will address antitrust only, mainly from a European perspective, but with some consideration of the US.

Antitrust is a collective name for different types of legislation. It includes at least prohibition of *cartels* (agreements between undertakings and concerted practices), of *abuse of dominant position* or 'monopolisation', and *merger control.* These areas of antitrust differ considerably. Cartels require the coordinated effort of more than one undertaking, whereas abuse of dominant position is in principle a practice committed by a single undertaking. Enforcement of cartel and abuse of dominant position rules mostly apply *ex post facto*, when the illegal practices have displayed some of their effects. Conversely, merger control happens *before* a merger between two undertakings has become effective, requiring notification of certain deals

Carlo Piana, *Antitrust, Competition, and Open Source* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Carlo Piana 2022. DOI: 10.1093/oso/9780198862345.003.0017

before they are consummated, therefore it is mostly *ex ante* and it requires a notable amount of forward-looking evaluation by the antitrust authorities.

It is not self-evident—or perhaps it *has not been* evident—why antitrust should be an area of concern for Open Source, if not when projects and businesses are on the receiving end of anticompetitive practices. After all, in the software industry, we have observed large and wealthy corporations thriving by monetising software in a proprietary fashion; Open Source has for a long time been seen as only a vague competitive threat to a collective dominance of this side of the software industry. This is the case of the two-decades-long antitrust actions to limit Microsoft dominance—and abuse thereof—in the PC operating systems market. After a long ineffective line of actions, agreements, new threats, and repeated abuses, the competitive landscape in some markets consisted of Microsoft dominating and other firms left without any hope of posing significant competitive constraint where it mattered (and most of the time depending and relying on Microsoft's own platform one way or the other). In the meantime a few Open Source initiatives resisted and gained some traction in the very markets Microsoft dominated, and by their nature could not be easily removed. Open Source as such was expressly targeted by anticompetitive and FUD[1] tactics. Eventually, Samba—an Open Source implementation of certain Microsoft networking protocols—was one of the driving forces that succeeded in a litigation following through a decision of the European Commission finding Microsoft liable of abuse of dominant position.

On the other hand, Open Source-making firms and projects have been put under antitrust scrutiny only to a limited extent but nonetheless cases have existed. Almost unnoticed, there has been at least one attempt to accuse Open Source projects of price fixing and predatory pricing. The same accusation was moved in a more widely known antitrust initiative against Google's Android operating system, an implementation of Linux (probably the largest and most successful Open Source project ever) for mobile devices.

In another case, the spotlight was put on the merger between two major software firms when Oracle notified the European Commission of its intention to merge with Sun Microsystems, the steward of four important Open Source projects, including the very successful and widely used database engine MySQL.

17.2 Abuse of Dominant Position

17.2.1 Concept

There is no precise legislative definition of when a firm holds a dominant position. Article 102 of the Treaty on the Functioning of the European Union (TFEU)[2]

[1] FUD: fear, uncertainty, and doubt.

[2] Any abuse by one or more undertakings of a dominant position within the internal market or in a substantial part of it shall be prohibited as incompatible with the internal market in so far as it may affect trade between member states. Such abuse may, in particular, consist in:

does not go any further than proclaiming that abusing a dominant position is prohibited, and does not give any further detail of what is dominance and when it is abused. *Dominance* is frequently associated with *market share*: the higher the share in a market, the more likely the entity has dominance. In essence, dominance is found when a firm is in a position which shields it from effective competition from other firms and enables it to act independently of its customers.[3] In other words, a dominant firm has considerable power stemming from its market share and allowed by the overall structure of the competitive landscape, the structure of the market, the control it exerts over certain facilities, the existence of barriers to entry, and the absence of alternative goods or services in adjacent markets which could fulfil the same needs of the customers.

Achieving and maintaining dominance—or even a monopoly—*is not prohibited*. Dominance imposes *additional obligations* upon the dominant undertaking in terms of fair competition. For instance, unlike any other firm, a dominant company could be required to deal with a competitor under fair and reasonable conditions when it holds an essential facility without which a competitor cannot effectively compete and which is not available otherwise. In the Microsoft case, this essential facility was found in certain *de facto* client–server and server–server network protocols, which it was required to license to competitors.

17.2.2 Predatory pricing and the Android case

Predatory pricing is a practice whereby a dominant company *trades below cost* with the *intent* of marginalising and excluding competition. The traditional theory claims that once the competition has been effectively excluded, the dominant company can then recoup its losses by raising its prices, and therein lies the anticompetitiveness and abusive nature of the practice. In other words, the savings for the consumer are only temporary, then negatively compensated by the higher prices that follow in a monopoly or in a heavily dominated oligopoly.

However, the recouping of the costs with higher prices at a later stage when competition is eliminated is only the most evident possible reason for predatory

(i) directly or indirectly imposing unfair purchase or selling prices or other unfair trading conditions;
(ii) limiting production, markets, or technical development to the prejudice of consumers;
(iii) applying dissimilar conditions to equivalent transactions with other trading parties, thereby placing them at a competitive disadvantage;
(iv) making the conclusion of contracts subject to acceptance by the other parties of supplementary obligations which, by their nature or according to commercial usage, have no connection with the subject of such contracts.

[3] See the judgment of the European Court of Justice in *Hoffman-La Roche*,. (*Hoffmann-La Roche & Co. AG v Commission of the European Communities*. Dominant position. Case 85/76).

pricing. The legal theory does not require this particular intent to find abusive behaviour. Another possible reason is eliminating potential competition and raising barriers to entry in an adjacent dominated market ('tying' is a similar case, when two products are sold together so that the second product or service comes for free together with a dominant one). The *required intent*—which needs not to be subjectively proven—is directed to the *elimination of competition through* excessive reduction of prices. An actual loss on marginal cost is neither sufficient nor required, and the simple intent without an excessive reduction of prices is also irrelevant; for instance, it is possible to charge a very low price if—due to synergies and scale economy that is unavailable to competitors and not obtained or maintained through other abusive actions—that price is still remunerative.[4] Below-cost sales are also considered irrelevant if maintained only for short terms (e.g. 'sell-offs').

The discussion of *how* a price can be considered abusively low is interesting, but beyond the scope of this book. Zero is a price that many would consider below any admissible threshold. Despite this, there is no indication that the price of Open Source software must always be zero; the rights granted by Open Source licences and the near absence of distribution costs make it unlikely that anybody would pay more than zero for acquiring a further copy of the software they have already obtained. But is the zero price unjustified and would a zero-priced copy of Open Source software be labelled as predatory? Such a claim has been attempted in the past—and failed.

The first reported case was brought against the Free Software Foundation (FSF), an organisation central to Open Source, by Daniel Wallace. The case was dismissed on preliminary grounds, on the basis that the plaintiff failed to properly prove the claim of harm to competition.[5]

The second case worth mentioning is of a much higher profile and had a somewhat more ambitious target: Google's Open Source operating system, Android.[6] Android is an Open Source project, at least at its roots. Built on the top of the same kernel as GNU/Linux (i.e. Linux), Android as a distribution is made freely downloadable from a public Open Source project named AOSP (Android Open Source Project).[7]

[4] See Moritz Lorenz, *An Introduction to EU Competition Law* (Cambridge: Cambridge University Press, 2013) 230–2.

[5] See *Wallace v Free Software Foundation Inc.* (case no. 1:05-cv-00618-JDT-TAB) <https://www.courtlistener.com/docket/4633603/wallace-v-free-software-foundation-inc/> accessed 18 April 2022. The order states: 'Mr. Wallace's alleged injury relates only to his personal inability or unwillingness to enter into the software market because his efforts might not be rewarded financially. This injury constitutes harm to Mr. Wallace as a competitor, not harm to consumers specifically, or harm to competition in general.'

[6] The author has had access to the complaint of FairSearch, but cannot discuss its particulars here as its distribution was limited and covered by procedural secrecy. No copy of it has been made publicly available.

[7] <https://source.android.com/> accessed 18 April 2022.

The claim was brought—amidst other apparently better-founded claims, and was the first to expressly target the mobile platforms—by a coalition of firms claiming to be affected by abusive conduct of Google both in the mobile software sector and in the search engine business, FairSearch.[8]

The only public disclosure of the claim in the Android case comes from the public statement of the complainant,[9] and it concentrates more on the 'bait and switch' strategy (i.e. releasing as Open Source software, but offering a more proprietary version to the original equipment manufacturer (OEM) marketplace) and fails to discuss the basis of the predatory behaviour in the case. It was mentioned in a laconic period in another press release.[10] The specialised press reported it[11] and other parties, including the Free Software Foundation Europe (FSFE), commented negatively on this particular angle of the case.[12]

This part of the claim failed to impress the Commission and was not picked up, unlike other points made by FairSearch which made their way into a landmark decision issued by the Commission (the Google Decision).[13]

While failing to provide any more useful detail as to the alleged anti-competitive nature of Open Source, the Google Decision provides a very insightful view of how the complexity of antitrust issues is almost impossible to resolve with current legal instruments and theories in a real-life scenario, and requires a great deal of discretional and creative law-making by the Commission (as well as other antitrust authorities) and judges.

But from the Google Decision the importance of the *forking option*[14] granted by the Open Source licensing of the software emerges clearly. As in the Oracle Decision, discussed shortly, the competitive pressure of being able to fork and therefore the value of forks in providing a way out of lock-in is discussed.[15] Conversely, the parts of Android that are not Open Source offer a striking comparison and show the negative impact that this licensing regime has on competitiveness.[16]

[8] <http://fairsearch.org/> accessed 18 April 2022.

[9] <http://fairsearch.org/summary-of-the-fairsearch-complaint-to-the-eu-against-google/> accessed 18 April 2022.

[10] <http://fairsearch.org/fairsearch-eu-googles-android-trojan-horse-dominate-mobile-markets/> accessed 18 April 2022. 'Google's predatory distribution of Android at below-cost makes it difficult for other providers of operating systems to recoup investments in competing with Google's dominant mobile platform, the complaint says.'

[11] <https://arstechnica.com/tech-policy/2013/04/opinion-antitrust-complaint-against-android-is-an-attack-on-open-source/> accessed 18 April 2022.

[12] <https://fsfe.org/activities/policy/eu/20130729.EC.Fairsearch.letter.en.html> accessed 18 April 2022. The author assisted FSFE with the submission.

[13] <https://ec.europa.eu/competition/antitrust/cases/dec_docs/40099/40099_9993_3.pdf> accessed 18 April 2022.

[14] See discussion of forking at Chapter 22.

[15] Google Decision, paras 1038–1046: 'For the reasons set out in this Section, the Commission concludes that Android forks constitute a credible competitive threat to Google.'

[16] Google Decision, paras 1114–1154.

Therefore, the first two behavioural remedies in the Google Decision with regard to the licensing of Google Search and Google Play Store prohibit Google from discriminating against forks of Android.[17]

17.2.3 The Microsoft case and the fairness of RAND conditions

It is undeniable that there has been a lot of historical tension between the company making Windows or the Office suite, and the Open Source camp. All of the sudden, in 2003–2004, this tension materialised in court, when two comparatively small outfits, FSFE[18] and the Samba team,[19] decided to join a handful of companies and industry associations to side with the European Commission and defend its antitrust decision of March 2004.[20] The case has been a seminal one in forced licensing, following through a line of cases that started with McGill and continued with IMS Health later. The decision imposed a requirement to release complete documentation of certain network protocols, that Microsoft imposed as *de facto* standards, under reasonable and non-discriminatory conditions (RAND) in a timely fashion. More accurately, the case had another side to it, relating to the tying of multimedia software to Windows, but this second angle is totally irrelevant to our analysis. Technically, the Monti decision found an abuse of a dominant position in the workgroup server market for both server to server and client to server protocols for file, printing, and authentication services.

Despite it being considered one of the most important antitrust cases to date, the first part of this story has little to do with Open Source, but because of the strategic importance of having FSFE and the Samba team on the sides of the Commission, after the initial complainant (Sun Microsystem), one of the most impacted companies (Novell), as well as an industry association (CCIA), decided to withdraw from the case through settlement under monetary compensation and in one case (Novell) through a merger, eventually Open Source became a centerpiece of it. Open Source was barely mentioned in the final decision[21] that found almost entirely in favour of the Commission and upheld only one minor part of

[17] Google Decision, paras 1398–1399.

[18] <https://fsfe.org> accessed 18 April 2022.

[19] <https://www.samba.org/> accessed 18 April 2022.

[20] Case COMP/C-3/37.792 (<https://ec.europa.eu/competition/antitrust/cases/dec_docs/37792/37792_4177_3.pdf> accessed 18 April 2022). Microsoft also known as the 'Monti decision', from the name of the Commissioner to antitrust of that time, Professor Mario Monti. For a comprehensive analysis of the economics, see Nicholas Banasevic and Per Hellström, 'Windows into the World of Abuse of Dominance: An Analysis of the Commission's 2004 Microsoft's Decision and the CFI's 2007 Judgment' in Luca Rubini (ed), *Microsoft on Trial: Legal and Economic Analysis of a Transatlantic Antitrust Case* (Cheltenham: Edward Elgar Publishing, 2000) 47–75. Per Hellström was the lawyer defending the Commission in Court and Nicholas Banasevic was the lead economist in the case team.

[21] Case T-201/04 *Microsoft v Commission*.

the case, despite the importance of the knowledge that Samba disclosed in the case, which was instrumental in dismantling the arguments on the impossibility and danger of releasing Microsoft's protocols to make them available to competing interoperable implementations.[22] The implementation of this, conversely, has been the phase where the Open Source nature of the main 'competitor' to Microsoft's own implementation—in fact the only competing independent one—played a major role and posed the deepest questions from a legal and technical point of view.

Microsoft decided not to appeal the judgment of the Court of First Instance (now General Court), therefore case T-201/04 became final.[23] The Decision, even after the partial amendment imposed by the Court, still required behavioural remedies and an implementation that was difficult to bring to completion. The obligation was, in summary twofold: on the one hand, the documentation had to be prepared in a way that conveyed all the relevant information and permitted interested third parties to appraise the nature, content, and value of the information. In other words, it needed to be *timely, complete, and accurate*. On the other hand, the documentation—once ready—had to be released to all interested parties under RAND conditions.

The first part, disclosure of information, basically consisted of merely technical issues. Nevertheless, reportedly the documentation required many man-years work to create something retroactively that ought to be documented in the first place, but surprisingly it had not. Microsoft reported to the parties in the compliance process that it had to conceive a totally new system to collect and organise the relevant information. In addition, there was a lot of debate on what the scope of the disclosure was. In a nutshell, Microsoft claimed only the 'on-the-wire' protocols were requested; conversely the Commission (and the Decision) confirmed that the extent of release was to everything necessary to permit full interoperability (including internal states, if relevant).

The second part, conversely, presented unexpected challenges, including a special subset presented by Open Source. 'RAND', *per se*, means nothing. What competitors required was a full-fledged technology transfer agreement negotiated directly. But because the conditions ought to be non-discriminatory, individual negotiation was out of question. In a commercial negotiation, the free will of the parties is paramount: if a party does not like what it is being offered, it can simply walk

[22] A personal note of gratitude goes to the members of the team, in particular Andrew 'Tridge' Tridgell, Volker Lendecke, and Jeremy Allison, who supported the team the author led as counsel of record.

[23] Even before the case was decided on its merits, Microsoft had been slapped with a decision finding non-compliance with the Decision, after its first attempt to block its application was rejected in the interim case before the Court of First Instance, T-201/2004-R <https://ec.europa.eu/competition/antitrust/cases/dec_docs/37792/37792_2186_8.pdf> accessed 18 April 2022.

away. In this case it was not a free negotiation, it was more akin to a collective bargain process. A lot of negotiation effort was spent on what fee structure ought to be put in place and *how valuable* the protocol information was, thus what a reasonable fee could have been. To make things more complicated, there were two 'intellectual property' (IP) matters to be considered: patents and trade secrets. According to the jurisprudence and the law of the European Union (EU), patents bring a presumption of valuable technology, therefore they are in principle royalty-generating. Trade secrets, conversely, can be valuable because of their inherent innovation and contribution to technology, or simply because of their *strategic role of being a secret*. The Decision clearly stated that Microsoft was entitled to receive compensation for the disclosure of trade secrets only based on the value of the released information, separating the technical merit from the strategic value of foreclosing competition. Using Samba's words as argued in court, if the information was not secret because it was valuable, but it was valuable because it had been kept secret, Microsoft could not have been allowed to benefit from the very anticompetitive behaviour it was forced to put a stop to.

Considering Open Source, this situation brought an additional angle, which is common to the entire matter of interaction between (open) standards and Open Source. On the one hand, requiring *running* (or per copy) *royalties* defies the very nature of Open Source, that by its nature is freely available, not registered, and so cannot impose reporting obligations. On the other hand, speaking of trade secrets, once the secret information is properly implemented as Open Source, distributing properly commented *source code inevitably discloses* a lot—if not the entirety—of the implemented secrets. Microsoft's proposals included a secrecy commitment on the source code, which was considered by the Samba team to be a clear discrimination against their licensing model (GNU General Public Licence version 3 (GPLv3)). Furthermore, at least initially, Microsoft insisted that the licence must be a *full bundle of all the rights*; therefore, an approaching licensee ought to license both trade secrets and patents.

Microsoft retorted that the incompatibility with their proposal and the licensing model of their competitors was not their fault, that it was a self-inflicted problem, since the licensing chosen by Samba was a free choice which could not be attributed to Microsoft, and in the alternative Samba could have mixed proprietary and Open Source software. The Samba team argued that this line of reasoning was very convenient, as Open Source was the only remaining competition after all the proprietary players had been forced out of the market, showing that only Open Source had been resilient against anticompetitive attacks; therefore, excluding the only viable competition form the remedies was a self-denying proposition. As per the issue of bundling, a competitor could have decided it did not need a licence under the patents, for many different reasons, including that it could have sold the implementation where patents were not valid; or it could have decided to challenge them in court if enforced against it; or it could try to invent around patents; or again a licensee could reserve to take a separate licence only after being successfully in building an *in vitro* implementation, for which patents are not relevant.

The Commission decided that Microsoft had both completed the documentation and offered RAND conditions in compliance with the Monti Decision only as of 22 October 2007. On that date Microsoft offered a licence without running royalties under trade secrets and an optional patent licence based on global revenues of the implementation. It also conceded that only partial confidentiality was required on the very documentation, but not on the implementation in source code. Consequently the Commission established a final date when Microsoft ceased to be non-compliant, and fixed the final amount of fines for non-compliance in further €899 million.[24]

However, the Samba team was not satisfied with the compliance decision and started a negotiation directly with Microsoft, although at this point Microsoft was not compelled to make further concessions. In the final agreement, signed on December 2007, it included partial assurances with regard to patents, using a 'speak up soon, or shut up completely' approach, meaning new hitherto unlisted patents could not be asserted against implementations of the licensed protocols if a complete notice was not given to the licensee as soon as the patented invention was implemented in a beta release of Microsoft own implementation.[25]

That was not the end, however. A further appeal was brought by Microsoft against the compliance decision, arguing that the concessions had been made under undue constraint by the Commission, which ought to have accepted Microsoft's conditions much earlier. Most of these further requests by the Commission, that according to Microsoft had unlawfully delayed a complete clearance, were those changes required to accommodate Open Source licensees. Therefore, again, the Samba team and FSFE teamed up to appear in Court as interested third parties and argued against the Microsoft line, in particular when it tried to diminish the role and rights of Open Source players and *ex post* rejecting the need to concede to them what it was forced to concede. Nearly ten years after the entire process started, on 27 June 2012 the General Court (the new name of the former Court of First Instance) found against the Microsoft appeal, only granted a slight reduction of the total amount of fines, but overall upholding the position of the Commission and indirectly that of the Samba team and of FSFE with regard to the licensing to Open Source.

[24] Decision C(2008) 764 final of 27 February 2008 <https://ec.europa.eu/competition/antitrust/cases/dec_docs/37792/37792_3997_9.pdf> accessed 18 April 2022.

[25] A clear and concise description can be found in Andrew Tridgell's blog post 'Samba and the PFIF' <https://www.samba.org/samba/PFIF/> accessed 18 April 2022.2022.

17.3 Merger Control

17.3.1 Concept

Unlike cartels—which are almost invariably secret and which may be made of implicit agreements for collusive practice to the detriment of competition—competing firms can agree upon very open and publicly disclosed agreements, which likewise could have permanent and structural impact over the market and reduce the overall competitive situation. This is the case of mergers and in general all cases of market concentration, which may include disposition of assets between two competing businesses.

Therefore, all transactions involving two or more competitors that exceed certain thresholds must be notified to the antitrust authorities well before being fully brought to completion, with disclosure of the relevant facts and figures. The antitrust authority must therefore assess if such deals are likely to raise antitrust concerns and need more scrutiny or conversely whether approval can be rubber-stamped.

When merger transactions involve global firms, potentially all jurisdictions could be involved, making clearance a very complicated issue. While formally, there is little difference between the US and Europe (as well as many other jurisdictions where a single transaction might be relevant), the outcome could be greatly divergent. In the US the Federal Trade Commission (FTC) is entrusted with merger control under the Hart–Scott–Rodino Act.[26] In Europe the European Commission is in charge under the Merger Regulation.[27]

The attitude towards mergers and their compatibility with a competitive market, and ultimately the desirable level of competition of a given market, is a matter of policy and economy, as well as a matter of legal assessment. In recent times—at the time of writing—a more liberal attitude from US authorities, more likely to allow for concentration of market power in few players can be observed compared to a more conservative attitude from the European ones, which are more likely to preserve diversity of players. The Oracle/Sun Microsystem—which will be discussed next—is one of those cases where this difference has been more strikingly noted.

17.3.2 Oracle/Sun Microsystems

Sun Microsystem can be considered an early adopter—among large and established IT-producing corporations—of the Open Source paradigm. During the

[26] <https://www.ftc.gov/enforcement/premerger-notification-program> accessed 18 April 2022.

[27] EC Merger Regulation no. 139/2004, see <https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32004R0139> accessed 18 April 2022.

last few years of its existence as an independent firm, Sun had an important portfolio of relevant and acknowledged Open Source products. Two of them resulted from 'open sourcing'—at least to large extent—their own flagship software products, namely *Java* and *Solaris*;[28] two of them were acquired—at different degrees of maturity—through a merger of existing projects led by independent firms. From the acquisition of StarDivision GmbH, Sun gained control of StarOffice, a competitor of Microsoft Office, which it renamed and published as Open Source under the brand 'OpenOffice.org'.

From the acquisition of MySQL AB, a Finnish operation in the business of Open Source database engines, Sun obtained control over MySQL, a largely adopted and successful database which became widely popular during the so-called dotcom revolution, because it provided the database component to the widely used LAMP platform and through this popular set-up permitted a large number of startups to thrive and rapidly grow between the end of twentieth century and the first decade of the twenty-first century.[29]

By the end of the first decade of the twenty-first century, it became apparent that Sun was approaching the last days of its independent life and was a potential target for hostile takeovers. Rumours of an imminent merger with IBM spread during 2010 and the early months of 2011. Eventually, merger talks apparently died off, and there was speculation of less-than-friendly takeovers. These were replaced by the rather unexpected announcement that Oracle Corp was the intended buyer of the Silicon Valley operation for the astounding figure of US$7.4 billion.

The deal was sealed in reportedly record time and the antitrust authorities received notification of it. The FTC, as widely expected, cleared the merger without any noticeable delay despite some Sun technologies could be considered relevant for the competitive setup of the market: the most valuable one was apparently Java. Because Java was used by many competitors of Oracle, and it was an important building block for software platforms running many workloads where Oracle was also ubiquitous, that was the single issue where attention was directed.

Very little attention was given to the other pieces: not on Solaris (which was relevant only in niche segments); not on OpenOffice.org (whose market share was negligible in front of the über-dominant Microsoft Office); not even on MySQL, which was thought incapable of scaling up and becoming a competitive threat to Oracle's own database, which in turn was considered too expensive to be viable for the workloads where MySQL was dominant. By everyone except Michael 'Monty'

[28] Sun Solaris is a Unix-like operating system. Java is a both a programing language and an environment providing libraries, a virtual machine for execution of code that is cross-platform by nature.

[29] LAMP stands for Linux, Apache, MySQL, PHP; respectively the operating system, the web server, the database engine, and the programing language that was used to build Internet-interactive websites and web applications on an Open Source stack.

Widenius, the founder of MySQL AB and 'father' of MySQL, who filed a complaint with the European Commission.[30]

The complaint by Mr Widenius was thoroughly considered by the European Commission, which opened a 'phase 2' on 3 September 2011, after two requests for information directed to competitors and database customers. This had the effect of stopping the clock for the merger, which was put on hold during this procedure. Eventually, after the phase 2 of the merger control case lapsed, the Commission issued a 'Statement of Objections', which is a formal document stating the case for a decision finding against the merger, allowing the concerned companies and all the interested parties to present their views and be heard in an impartial, finely regulated, administrative procedure.[31]

Indeed, the Statement of Objections, a rather voluminous document consisting of in excess of 150 pages, provided among other—for us—less interesting parts a somewhat insightful description of the economics behind Open Source projects, manly of the same nature as MySQL, which was Open Source licensed (its main licence being the GNU GPLv2 with some linking exceptions), but not run as a 'typical' Open Source project. MySQL in terms of development was developed in a 'silo' fashion, with one single company holding the entirety of its copyright, which it exploited in a 'dual-licensing' scheme. Eventually the case ended with a decision finding the merger compatible with the market, after a hearing was held (Decision of 21/01/2010 in Case No. COMP/M.5529—*Oracle/Sun Microsystems*, hereinafter Oracle Decision).[32]

One of the major points that Oracle used to respond to the competitive concerns of the Commission was that because MySQL was indeed totally licensed as Open Source, the incentives for Oracle to thwart its success and its alleged likelihood to pose a competitive threat to Oracle's own product were largely overstated. Oracle's argument was that in the case of its becoming a poor steward of the project, the *threat of a fork* was sufficient to stop any such scheme, because a competitor, or collaborative project, could pick it up and become almost overnight a more viable alternative—something that happened with OpenOffice.org, which in a few months became a recessive project compared to LibreOffice, a community fork of it. Indeed, Oracle pointed out that a few forks of MySQL already existed, including the one started by Monty Widenius himself (MariaDB).

Widenius and the supporting parties to his position retorted with a rather solid point, at least in theory, that deserves attention. They claimed that a company or a

[30] The entire case file (non-confidential) can be found at <https://ec.europa.eu/competition/elojade/isef/case_details.cfm?proc_code=2_M_5529> accessed 18 April 2022.

[31] I have personally been retained by the lawyers advising Oracle at this time, but all information I provide herewith comes from public sources and I will try to provide a dispassionate recount of what I have partly personally witnessed.

[32] <https://ec.europa.eu/competition/mergers/cases/decisions/m5529_20100121_20682_en.pdf> accessed 18 April 2022.

community not owning the copyright over the project might not set up a credible fork against the copyright holder as it would be at competitive disadvantage being unable to obtain the proceedings from the 'proprietary' licensing of the forked project, as MySQL's owner, being left with less rewarding sources of financing as 'support services'. While in theory this was valid, the point falls flat on its face as there was—and in general there still is—no evidence that for a project to be successful and resourceful it needs to be able to exploit code in a proprietary fashion.[33] The existence of several projects in similar fields that do not rely on proprietary exploitation can be considered convincing counterfactuals.[34] Whether or not this claim was well founded, the fact that this could indeed have been convincing if proven is relevant in our analysis. While it cannot be argued that control over copyright is a necessary precondition for *any* project to be successful, on the one hand there *might be* projects that need such control and therefore a fork would be gravely disadvantaged and could be no competitive threat; on the other hand, it is apparent that projects which are *currently* not owned by a single entity, but are successful none the less, are more likely to also be successful if forked. If that was the case, a merger of its principal steward with a competitor would not pose the same competitive challenges as if the steward was also the sole copyright holder or relied on proprietary exploitation of the solution or of parts thereof. Open Source is therefore demonstrated to be a relevant issue for merger regulation issues. But more importantly control of copyright (or lack thereof) could be far more relevant.

The investigation came to an end when Oracle published certain unilateral pledges made to solve the antitrust concerns of the Commission, and the Commission published a decision where it found the merger compatible with the market in the light of the commitments. Despite being favourable, the Oracle Decision is severely critical towards the merger and goes to great lengths to describe, for instance, why a fork of MySQL would *not* have been successful.[35]

[33] The Oracle Decision discusses the IP implications of a fork, including this very argument, in paras 715–749, which is less than convincing as it poses too much emphasis on proprietary exploitation, which reflects a decade-old analysis coming from even older view of what constitutes an incentive to invest and whether it is feasible to invest in 'pure' Open Source initiatives, as if GPL-only projects could not attract relevant investments from independent companies, like Linux quite evidently shows. In a way, the Commission was right, because it looked at MySQL as a non-pure Open Source project, but it failed to see what a fork, as a pure Open Source project, could have become. In all honesty, the Commission only had a few instruments and not much experience to rely on for a forward-looking analysis, and it relied on 'conventional wisdom', rather than the one—in hindsight correct—proposed by Oracle, which included the supporting document by Professor Eben Moglen, mentioned in footnotes 457 and 459 of that Decision. The Decision in the IBM/Red Hat case, which we discuss later in this chapter, is much more informative and forward-looking in such respect.

[34] A counterfactual is in the economic analysis, a real-life evidence that can prove (or disprove) a claim by showing that if the logical premises of such claims were right, such examples would not exist. In this case, the fact that numerous examples of thriving projects without proprietary control of copyright would disprove that this is a requirement, unless other elements were added. In the absence of *in vitro* controlled experiments—which are at least impractical in economy—the search of a counterfactual in the next best solution to prove an economic theory.

[35] See paras 678–750 in the Oracle Decision.

Personally, I believe that many of the concerns have been overestimated. However, the decision remains a very useful basis for any analysis of the economic dynamics involving an Open Source project both from an antitrust perspective and in general.

The decision of the Commission was challenged in court, but the case has been withdrawn from the docket before reaching the oral phase.[36]

17.3.3 IBM/Red Hat

As history repeats itself but never in the same way—and sometimes with strikingly different outcomes—another merger came to relevance in more recent times. When IBM Corp. announced its intention to merge with Red Hat Inc., in 2018, many saw in it a potentially controversial and heavily litigated merger control case, similar to what happened a few years before with Sun/Oracle. The ingredients were all there. Red Hat is by far the largest and most successful full Open Source company. It is steward of some of the most popular Open Source projects, sometimes having the largest share if not the entirety of the copyright of said projects. IBM and Red Hat, in addition, might have many more points of overlap than Oracle and MySQL had in the previous case. They are in the same space both for workloads and intended clientele (including the lucrative and more concentrated high-end sector of many markets, chiefly middleware). Size-wise, this transaction is almost five times larger than Sun/Oracle.

The current doctrine of mergers states that vertical mergers are, as a general rule, pro-competitive, while horizontal mergers are more likely to be anticompetitive. Larger transactions may be difficult to fit into only one of these two categories, as they have elements of both. IBM and Red Hat have different offerings. In particular, Red Hat is an important business partner of IBM with its Linux distribution and related services. In addition, their middleware services occupy the same space. Potentially, after the merger (actual and potential) competition became significantly lowered, what would the options be? In similar cases, there are two main alternatives: assessing the entire transaction as a whole and weighing the preponderance of effects, whether they are pro- or anticompetitive and clear or prohibit the transaction as a whole; the alternative could be to consider that the pro-competitive effect prevails, but decide that the anticompetitive part is also important and has effects which are incompatible with the market, therefore clearing the transaction *under condition of divestiture*.

[36] Case T-292/10 *Monty Program AB v Commission*, <http://curia.europa.eu/juris/document/document.jsf?docid=78981&doclang=en> accessed 18 April 2022. The author was counsel of record for one of the intervening parties and cannot further discuss the particulars.

When the business is mainly based on an Open Source product which you develop in a *fully collaborative environment*, what does *divesting an Open Source business entail*? Would it imply a prohibition for the divesting entity to trade the product or the services? Divestiture in general means giving assets to an independent firm that will develop the business externally to the firm resulting from merger. But what is 'business' in an Open Source project developed in the open, and the divesting company is only the major corporate steward of it? As frequently happens in Open Source, common wisdom is reversed: forcing divestiture of an Open Source project would achieve counterintuitive effects.

First and foremost, what would the divesting company sell? Copyright? That would mean little, if the work is of a collective nature and no copyright holder can separate its contributions from the others. Trademarks? However, a product trademark associated to an Open Source project would have less intrinsic value and the interactions with the project's trademark policies would bind the hands of the new owner anyway. A service trademark, conversely, would probably mean also divesting support services, personnel, contracts, and platforms. Patents? In case of Red Hat there were a few patents, but Red Hat pledges to put those patents at the service of the communities and has a general non-assertion policy, which is presumed to be enforceable against Red Hat itself and successors thereto.

Divesting an Open Source project leaving it without strong corporate support is likely to mean making it less worthy for corporate use. If the project thrives anyway, it probably means that the competitive threat would have been higher with an additional supporting entity. In the case of Red Hat, it is difficult to separate how the product is worthwhile or its value from the reputation of its corporate steward. Even if the project was assigned to an independent foundation, would this attract sufficient corporate sponsors and development energy to make it an equally viable project for business use?

Eventually, despite expectations of a second-phase procedure by the Commission, the merger was cleared without any further requirements, after receiving a similar unconditional clearance by the FTC. With reference to the overlapping middleware services offerings, it was found that existing and potential competition would have been significant unaffected. The Open Source nature of Red Hat's offering and the potential reaction of the community (and of developers) should the merged entity decide to degrade their support to the project, have conversely had a central role in justifying an immediate clearance, which went at length analysing the competition aspects of Open Source projects and is a valuable source for understanding further.

For a summary of the case see <https://ec.europa.eu/competition/elojade/isef/case_details.cfm?proc_code=2_M_9205>. The assessment of the Commission reads, in a nutshell, as follows (para 569, p 122):

> Overall, the Commission considers that the Merged Entity will not have the ability and incentive to delay the development of Open Source projects it is involved in or otherwise redirect them to reduce the emergence of competing products or reduce the competitive pressure of existing products. This is based on the following general reasons and on the specific reasons set out below at paragraphs (583) to (602).

The decision can be found at <https://ec.europa.eu/competition/mergers/cases/decisions/m9205_1200_3.pdf> accessed 18 April 2022.

18
Foundations and Other Organisations

Karen Sandler

Free and Open Source software represents an impressive coordination of contributions of many people across different geographies, cultures, and motivations. Over the years, Open Source communities have forged mechanisms to facilitate this collaboration that codify priorities and ground rules that work together. Often, these mechanisms are embodied as legal organisations around major Open Source initiatives. This chapter will explore the main types of foundations that serve to forward Open Source.

The term 'community' is used in many ways in Open Source, often without precision. Companies will talk about how they 'give back to the community' or expect 'the community will contribute' when code is thrown over the wall and published as a *fait accompli*. People hold jobs with titles like 'community manager' who don't actually manage anyone in the loosely defined group of people comprising employees of other companies as well as volunteer hobbyists. Indeed, many of the terms used in Open Source are similarly imprecise and used in ways that only promote the agenda of the speaker. Even what comprises an Open Source 'project' is up for interpretation and could mean work on a narrow stand-alone piece of software or it could mean efforts on diverse pieces of software that are loosely connected. Here, we will use the term community to refer to any group of contributors self-organising into a group[1] and Open Source project to refer to the software that is the subject of that effort.

[1] The use of the term 'community' here is informed by the use of the term by the Outreachy internship program, a diversity initiative that provides paid, remote internships to people subject to systemic

Karen Sandler, *Foundations and Other Organisations* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Karen Sandler 2022. DOI: 10.1093/oso/9780198862345.003.0018

18.1 Governance versus Foundations

While this chapter focuses on the legal structures seen in the Open Source world, those structures emerge from the relationships established early in a community's collaboration. The choice of legal structure or foundation can embody the ideals and principles of the underlying community and define the rules of engagement, but often those relationships establish more organically. Some communities make their decisions by informal consent, others by following a charismatic leader,[2] some by adopting more formal majority decision-making, and still others by giving authority to those contributing the most. These relationships are discussed in Chapter 2 and often morph into an informal governance structure that may or may not be reflected in the legal construct that is overlaid by a foundation.

The community's governance could be identical with the corporate governance of a legal structure or it may be largely unrelated. Some communities have specifically formed foundations where the technical direction of the project is divorced from the foundation. While a foundation can successfully fill a financial and other supporting role to the project without steering its technical direction, models where the foundation operates with a distinct governance structure from the technical governance can lead to tension in the community. Those foundations can be ineffectual as the organisation uses its resources without a focus on the major technical problems of a project. Organisational leadership often lacks the respect of the community at large, which impairs its ability to build relationships and enhance collaboration. Conversely, a project can have a successful governance model independent of any legal structure. Open Source projects by nature value transparency and are predicated on the cooperation of diverse participants, so many projects organically develop functional governance before any legal foundation is contemplated.

These issues tend to be more complicated for projects that originate in for-profit companies where the interests of a founding corporation to retain control of the project must be balanced with its desire to work with others, receive contributions, and develop a community. In those circumstances, governance is the product of negotiation between prospective business partners and can be more akin to a Memorandum of Understanding. The economic drivers which impact this have also been discussed further at Chapter 15.

bias and impacted by underrepresentation in the technical industry of the country they are living in. Outreachy is a part of Software Freedom Conservancy.

[2] Sometimes referred to as a 'Benevolent Dictator'.

18.2 The No-Foundation Solution

Other than those that are products of for-profit companies, most Open Source projects begin without any legal structure at all. These projects start with a public repository and a forum or mailing list for conversation about important topics in the project. Depending on how the community is composed, it can proceed without any legal organisation for quite some time. Whereas some foundations once made it their key goal to provide critical infrastructure for their projects, the reliance on cloud services has made this much less necessary. The biggest pitfalls of the no-foundation solution relate to a lack of coordination around trademarks and other intellectual property choices. In particular, there have been several cases where a single member of a community filed the trademark of a project as an individual and subsequently used that trademark to brand a commercial enterprise without permission or cooperation of the rest of the community. Because of the absence of a neutral entity to hold the mark, even though the activity went against the community's informal governance and understanding of how they collaborated together, there was no legal recourse.

A lack of a foundation around an Open Source community may not indicate a lack of robust formal governance or a lack of legal organisation to hold some of the rights of a project. An example of this is the Debian project. Debian has over a thousand Debian Developers (DDs) who are active contributors who have applied for special voting status in the community which requires passing a comprehensive test about the values of software freedom, the expectations around collaboration in the Debian community, and even about aspects of Open Source licensing to achieve that DD status. DDs select a Debian Project Leader (DPL) on an annual basis who formally represents the community and appoints the Technical Committee and FTP Masters, which make important technical decisions for the project. While Debian affiliates with three organisations, one in the US, one in France, and one in Switzerland, to accept funds and handle some legal issues, there is no single foundation that represents the project as a whole and the governance is wholly divorced from any legal organisation.

18.3 Charities

One of the oldest forms of foundation chosen by Open Source projects is charities, organised for the public good. With no shareholders and a mission to serve the public, charities are a way for projects to organise and ensure that no company or individual can take advantage of the community. Choosing a charitable foundation signals a commitment by the community to keep an even playing field among individual contributors and corporations, big as well as small.

The Software Freedom Conservancy, GNOME Foundation, and ASF are all examples of organisations in this model. For charities based in the US, like the ones listed earlier, the legal structure involves first a state incorporation and then recognition by the federal Internal Revenue Service (IRS) of the relevant statutory tax exemption for charities, referred to by its section in the US Tax Code, section 501c3. Within the US, 501c3 organisations must file annual reports that are made public by the IRS and which disclose critical financial information about the organisation. Outside of the US, organisations are also formed under comparable charitable legal rules like, for example as a *Stiftung* in Germany, like the Document Foundation.[3]

Charitable organisations are generally split into two basic categories: ones that are organised with a voting membership and ones that rely on a self-perpetuating board of directors.[4] Many of the early Open Source organisations adopted the membership model, which can bestow voting status on contributors to the project. Members are able to run for board seats, become officers and participate in the elections to appoint those roles. For many of these organisations, members can earn their status through contributions made in the course of their employment, but their membership status is expected to be in a personal capacity and not transferable to another employee at their company. While most Open Source organisations prioritise transparency and publish board meeting minutes and other foundation documents, members often have private mailing lists where they discuss organisational or other sensitive issues that may arise for the project.

There are clear benefits to the membership approach. Creating a class of members rewards and elevates regular contributors in the community which encourages continued involvement. It helps make sure that important decisions made on behalf of the community are made by people who are rooted in that community and also responsive to the concerns of current contributors. However, member-run organisations have clear drawbacks. Member-elected boards comprise people who are well-known contributors to the project but may not have any experience running a charitable organisation. They may not have the background to understand the organisation's legal obligations and they may have no experience with fundraising, public relations (PR), or any of the other activities boards must engage in to ensure the long-term success of the organisation. Board members are chosen from the members who have received the most votes, without taking into consideration the overall composition of the board to have a mix of skills and strengths (other than making sure that there is not a conflict of interest created by too many

[3] See <http://flossfoundations.org/foundation-directory> accessed 18 April 2022 for an inclusive list of Open Source organisations of all kinds.

[4] There are a variety of organisations that are organised with hybrid models. For example, the Open Source Initiative is a 501c3 organisation that has created a class of voting members that is not defined by the legal governance of the organisation. Whether the existing board accepts the votes of the individual donor base or the affiliate organisation is at the sole discretion of its self-perpetuating board.

board members with affiliation to the same employer). Often, the election process happens on an annual basis or otherwise so frequently that directors with no previous board experience are only getting the hang of the position when their term ends.

In contrast to the membership model, organisations with a self-perpetuating board form with a selected group of directors that accepts stewardship of the organisation. Those directors can create or fill vacancies on the board, change the foundational documents, allocate resources, and define the overall direction of the organisation. Some organisations with self-perpetuating boards have term limits on their directorships, but most do not. Self-perpetuating boards often have the advantage of being assembled thoughtfully, with a combination of members that provide useful skills to the organisation. A given board may include long-time contributors that have the deep respect of the community, along with other members that are selected for their profile outside of the community, their connection with wealthy donors or grant-making institutions, knowledge of non-profit management, or any other number of skills that may be important to a project. Self-perpetuating organisations often have the opportunity to perform regular assessments on their efficacy and can adjust their membership accordingly. Where there are no term limits, directors may serve on the board for many years, establishing a knowledge base, dedicated skillset, and a sense of history about the project and the organisation. A self-perpetuating board can bring long-term stability to a foundation and to a community and avoid drama that can surface during heated election periods.

As with member-run organisations, the strengths of the self-perpetuating board are also connected to its weaknesses. Self-perpetuating boards can become out of touch with the organisation's community and be ignorant of the current needs of the project. Because boards are generally composed of volunteers, directors may decrease the resources they dedicate to the organisation over time and treat the role as a *pro forma* obligation. These directors may not have been active participants in the community for years. Self-perpetuating boards are also more prone to be impacted by 'founder's syndrome', where a charismatic founder of the organisation retains a disproportionate amount of influence over its operations and leadership.

Some charities have a paid staff whereas others are entirely run by volunteers. Organisations that are volunteer run generally rely on working boards of directors. Charities that have staff largely pay their employees at a level of compensation that is substantially lower than other foundational models like trade associations that will be described shortly. As is the case in other charitable causes, these lower salaries are commensurate with an increased sense of public service and dedication to a public good.

In contrast with trade associations, charities are necessarily free from corporate influence, which allows and even mandates the foundation to use its resources best to do the most good. This often results in more power residing with individual contributors to the project rather than their employers, and can provide a stopgap against corporate focus on quarterly results, allowing communities to think longer term.

18.4 Trade Associations

Open Source organisations that prioritise the interest of for-profit interests generally form as trade associations. These organisations are formed to forward a common business interest and within the US are often recognised as non-profit 501c6 organisations. Examples of Open Source trade associations are the Linux Foundation and the OpenStack Foundation. Most of these organisations have different levels of corporate membership with the board of directors formed according to membership levels. Many use precious metal labels for these levels (platinum, gold, silver, etc.) where the top-level members each receive a seat on the board and the lower levels are able to elect representatives among members of their level,[5] so only a certain number of the lower-level members are able to hold a board seat. Some of these boards also include a minority of 'community' board positions that are appointed by the paying member directors.

Trade associations pave the way for smooth corporate participation in Open Source communities. They often fill an important role of helping to transition corporate Open Source projects to include a broader range of contributors and to increase adoption in the industry. Trade associations are well positioned to provide professional training courses and certifications. As corporate-driven organisations, they are often more polished in presentation and have access to substantial marketing and PR resources. Where charities are restricted in their activities and the benefits they can provide to corporate donors, trade associations are able to highlight their corporate participants and be directly responsive to their interests. Trade associations are also particularly well suited to run events that feature for-profit contributions to Open Source projects and to provide industry networking and collaboration.

Trade associations are sometimes criticised as being 'pay to play', where participants are only able to maintain influence at the level at which they are able to contribute financially to the organisation. This can have the effect of prioritising the interests of the top-level platinum members to the detriment of smaller players.[6]

[5] Charities sometimes also use precious metal labels to indicate their sponsorship levels, though often use other terminology. For charities, the sponsorship levels are indicated in logo placement on organisational websites but do not reflect any corresponding control of the organisation via membership on the board of directors.

[6] These issues were highlighted in connection with ArduPilot's withdrawal from the DroneCode project. <https://lwn.net/Articles/700479/> accessed 18 April 2022.

18.5 Aggregating Foundations—Fiscal Sponsors

In both the charitable and trade association sectors, aggregating foundations have emerged to act as umbrellas for Open Source projects so that each individual community need not form a new legal entity. Modelled as fiscal sponsors, these organisations allow Open Source projects to join their legal entity which then provides financial, accounting, administrative, and legal infrastructure so that projects can receive funds, hold trademarks, sign contracts, run events, and perform a variety of other roles for the project. These organisations generally take a percentage of the project's incoming revenue to fund their operations. Examples of charitable fiscal sponsors are ASF and Software Freedom Conservancy.[7] Examples of trade association fiscal sponsors are Open Collective and OpenStack Foundation.

OpenCollective[8] is the most prominent example of a group of relatively new organisations that provide a more limited range of fiscal sponsorship services and do so in a more automated way, making it easy for Open Source projects to set up a way to receive and spend funds immediately. However, because the involvement of the organisation is limited, this approach may exacerbate problems in a community and can sidestep the establishment of necessary governance.

18.6 Corporate Initiatives

There has also been increased interest in corporate platforms to perform some of the roles that traditionally have been the purview of foundations. Companies have increasingly been hosts of Open Source projects, whether they be start up corporations funded by venture capital or Fortune 500 companies that keep a sponsorship and stewardship role in projects they establish or acquire. Platforms encouraging donations by the donating public have been established by corporations, a prominent recent example being GitHub Sponsors.

Depending on the long-term goals of the companies backing each initiative, the role of a single corporate player at the helm of a project may have a chilling effect on collaboration across the industry or may hamper the development of an independent community.

[7] This author is the Executive Director of Software Freedom Conservancy.

[8] There are multiple organisations affiliated with OpenCollective, including Open Collective Foundation, which is a charitable organisation.

18.7 A Note of Licensing and Foundations

While a subject for other chapters of this book, the licence an Open Source community chooses is fundamental to its governance and organisation. Early in the history of software freedom, ideological communities were split between those choosing copyleft licensing and those who preferred non-copyleft 'permissive' licensing. The two camps embodied different ideas of what freedom meant and whether it was more important that the software remain free in perpetuity or whether there should instead be freedom to create *any* derivative work, including proprietary versions. Over the decades that have followed, most for-profit corporations came to prefer and even insist on non-copyleft licences to reduce their ongoing obligations, retain more control over their software, and in many cases preserve the possibility of moving to a proprietary approach in future. The ideological community has largely shifted to prefer copyleft licensing, among other reasons, to protect against corporate interests. Many of the individuals in the early anti-copyleft camp have since come forward as now strongly preferring copyleft as a result of the fragmentation and problematic behaviour of for-profit participants.[9]

In 2015, Martin Fink, the Chief Technical Officer (CTO) of Hewlett Packard (HP), gave a talk at LinuxCon Europe, in which he described HP's choice to make the GNU General Public Licence (GPL) the company's default licence. In his talk, he described how a strong copyleft licence establishes a level playing field among stakeholders. He also made the point that without a copyleft licence, those stakeholders must establish governance through corporate structures that can be very expensive and time consuming. Generally, this has played out as Fink described. To prevent fragmentation and preserve a playing field that remains neutral enough that a broad base of contributors will participate, non-copyleft communities establish expensive boards of directors and invest considerable resources in their maintenance. Communities formed around copyleft licences can also have expensive foundational structures, but those structures have less at stake. Small players cannot be pushed out and ideological initiatives around the project are able to flourish, even for projects that have significant corporate interest and investment.

18.8 Co-option, Funding, and Confusion around Corporate Models

Despite the differences in the structure of the foundations in Open Source, charities, trade associations, and other organisations may all take in funds from a

[9] For example, Keith Packard spoke in 2020 about the political history of the X Windows System and how it shaped his views on copyleft. <https://www.youtube.com/watch?v=cj02_UeUnGQ> accessed 18 April 2022.

variety of sources. This includes corporate donations and donations from individuals. Unlike most other fields, successful Open Source projects that are founded for ideological reasons for the public good may also be useful to corporations that donate to ensure the sustainability of software they rely on, to garner goodwill from the community with an eye towards talent recruitment or simply for PR logo placement. Corporate-driven projects may inspire individuals to donate to them and offer donor status programs for individuals that use the same terminology as is traditionally used in the charitable sector. Corporate initiatives have in many cases more fundamentally co-opted the rhetoric of the ideologically driven communities and describe their products as 'making the world better'[10] and their actions as 'the democratisation of code'. This has led to some confusion about where the for-profit interests in the software and the non-profit ideological movement begin and end. Both trade associations and charities are 'non-profit'. Consequently, many long-time Open Source contributors do not understand how different Open Source projects and their foundations are structured, and how that influences the way in which they can use their funds and other assets. Because Open Source communities are global, many donors are not in the same country that the organisation they are giving to is situated and so do not expect to benefit from tax-deductible donations to charities. While many organisations follow IRS rules strictly and act fully in accordance with their organisational model, there are Open Source charitable organisations that have not been as diligent and have allowed corporate interests to drive some of their activities.

Perhaps it was this confusion and obfuscation that resulted in the US IRS putting 'open source' on a 'BOLO' or 'Be on the Look Out' list of words and phrases that triggered a disproportionately aggressive review by the agency. Flagged organisations found their applications subject to additional documentation requests and stalled without any feedback on submissions over long periods of time. The BOLO list primarily included political entries, like 'tea party' and 'occupy'. It also included 'non-profit journalism'. The subsequent scandal prompted by the political nature of most of the BOLO list resulted in an apology, investigation, and several resignations from IRS leadership. Some Open Source organisations that had applied for 501c3 recognition waited for many years with no response.[11] In addition, at around the same time there was also increased scrutiny of 501c6 applications, including the prominent rejection of the OpenStack Foundation, subsequently approved on appeal. Since that time, the IRS has instituted different review processes and many

[10] The television show 'Silicon Valley' captured this phenomenon powerfully throughout its first season, with one if its fictional giant tech companies having the slogan 'Hooli is about making the world a better place, through minimal message-oriented transport layers'.

[11] See the description of the Yorba Foundation's IRS refusal of 501c3 tax exemption at <https://lwn.net/Articles/604885/> accessed 18 April 2022.

Open Source charities that clearly articulate their connection to charitable work have been recognised by the IRS. While the long delays and uneven scrutiny were clearly inappropriate, it is not surprising that the IRS struggled with evaluating Open Source organisations.

18.9 Need for Organisational Diversity

While Open Source foundations in many cases provide overlapping services in overlapping niches, each existing organisation is tailored to particular communities and particular needs. With a variety of contributors to Open Source, all with a variety of motivations and goals, it is essential that a wide range of organisations exist in the sector. Open Source has never been more critical, whether to industry or to society. As the number and importance of Open Source projects increases, so does the workload to support those projects and help them thrive. As many foundations as we already have to serve this supporting road, we will need still more organisations and increased resources given to existing organisations to sustain the inevitable growth of Open Source in the years to come.

19
The Rise of the Open Source Program Offices (OSPO)

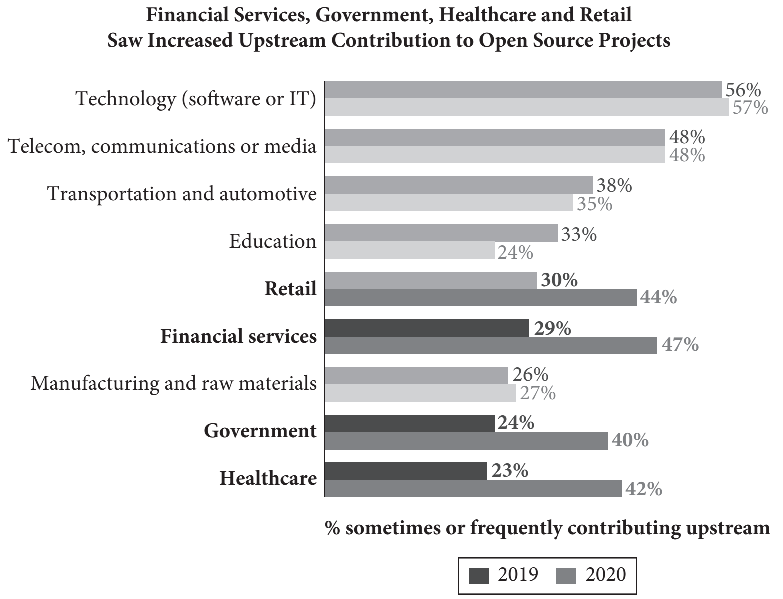
Nithya Ruff

19.1 The Beginning

The coming together of the Linux Kernel released by Linus Torvalds in 1991 with the GNU Tools (created in 1984) started the GNU/Linux Operating System (OS). This became a viable alternative for proprietary operating systems and began the era of widespread Open Source use as is further discussed in Chapter 1. The General Public Licence (GPL) discussed in Chapter 4 has been widely adopted and changed how people licensed software. This began a new era in how a significant amount of software is created and consumed. Linux has become an underlying operating system of choice for many products and industries—from the supercomputer to the phones in our hands, and more. And the Open Source way of developing and releasing software has now touched components and technologies in every single area, from cloud to embedded software and even hardware and data.

Fast forward to 2021, and it is hard to find organisations or enterprises that do not use Open Source in some way (see Figure 19.1). Technology companies, in fact, have been using Open Source since the early days. Today, even very traditional industries and governments use Open Source software. What started as cost-savings efforts by systems administrators is now intentionally and proactively used for its ability to accelerate innovation. Digital transformation has

Nithya Ruff, *The Rise of the Open Source Program Offices (OSPO)* In: *Open Source Law, Policy and Practice*.
Edited by: Amanda Brock, Oxford University Press. © Nithya Ruff 2022. DOI: 10.1093/oso/9780198862345.003.0019



Source: "Open Source Programs in the Enterprise - 2020" Survey.
How often does your company contribute upstream? 2020, n = 669; 2019, n = 1644, 2020 Industry verticals shown in chart: Education, n = 42; Manufacturing and raw materials, n = 26; Transportation and automotive, n = 31; Government, n = 25; Healthcare, n = 31; Retail, n = 18; Financial services, n = 49; Telecom, communications or media, n = 62; Technology (software or IT), n = 265.

Figure 19.1 Wide Variety of Organisations Contribute to Open Source

skyrocketed the scale of Open Source, in terms of usage and reach. Companies tend to go through a pattern, from usage to contributing to existing projects and open sourcing their own code. This extended usage has awakened a wide range of organisations to the value of hiring and supporting Open Source developers. From legal to human resources, marketing, finance, and customer care (and everything in between), savvy department heads and business owners began to manage all aspects of Open Source more actively and strategically. From the start of this century, we saw the rise of the Open Source manager role in companies using Open Source software.

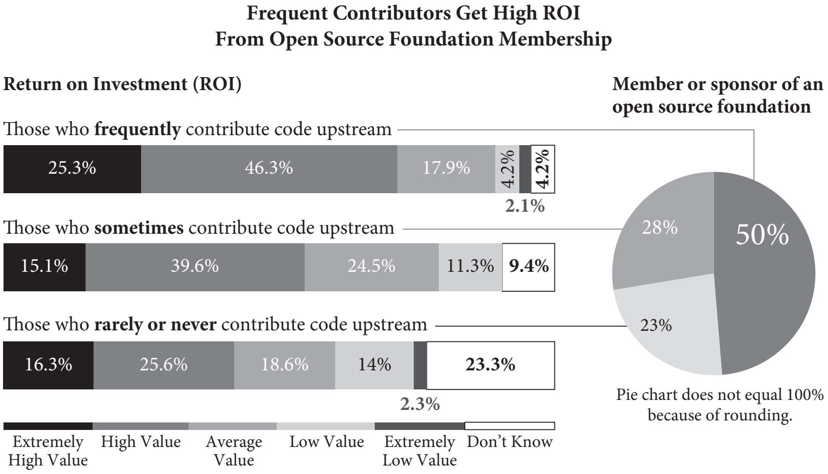
19.2 Should You Start an Open Source Program Office (OSPO)?

OSPOs have become *de rigeur* from 2020 and many organisations wonder how they benefit from starting one and, perhaps, why they should invest in one, especially when they are already consuming Open Source software. What is the advantage of engaging, beyond just downloading and using the code? This is a valid and inevitable question. The biggest reason to do so is to drive product/service intention and strategy proactively, which, incidentally, happens to be the biggest benefit of engaging with Open Source projects. Proactively driving product/service strategy tends to speed time to market, providing a competitive advantage.

What does this mean? When you ask engaged companies where they derive the most value from their own Open Source engagement efforts, they often point to projects that are core to their products—projects that let them collaborate with industry peers, learn, and share their stories, experiences, and lessons through various Open Source foundations and their events and activities. By contrast, opting simply to consume Open Source software without engaging in the process and the community can place an artificial limit on how much value organisations can realise from that Open Source software and their ability to influence.

Here are some drivers that may lead organisations to support the development of an OPSO:

- Consumption without proper understanding and internal governance can introduce risk to a company, such as having to depend on poorly maintained projects, navigate changes in direction, or use certain licences that may be problematic.
- Consumption of Open Source resources without contribution can result in the build-up of steep technical debt. This translates into increased reliance on forked software versions, which an organisation would have to support and integrate in-house. Forking can cause users to fall behind on security patches and miss out on ongoing innovation in the mainline project.
- Consumption without contribution means an organisation has little voice in the associated Open Source communities. This matters, especially when it may benefit an organisation to steer a project in a new direction or add important features that can be highly beneficial to the business and project direction. Contributions can be strategic, allowing a company to introduce new features and ways of doing things into the technology stack that can be put to good use elsewhere.
- With Open Source, we rely, to an extent, on external development teams. Failing to engage with these teams means giving up on the ability to manage and direct a key development resource.
- Without a systematic compliance plan, there is a risk of teams not following licence obligations and other good governance practices, which opens the door to legal issues.
- Perhaps most importantly, leadership in Open Source simultaneously brings both visibility and engagement. It results in a stronger and more credible technology brand, which makes a company more attractive in terms of recruiting and keeping great developers.



Source: "Open Source Programs in the Enterprise - 2020" Survey.
Is your company a member or sponsor of an open source foundation(s)? (e.g., Linux Foundation, Apache Foundation, Eclipse Foundation, Open JS Foundation) If yes, how valuable is the support and return on your investment you have received from these open source foundations? Frequently contribute code upstream, n = 95; Sometimes contribute code upstream, n = 53; Rarely or never contribute code upstream, n = 43.

Figure 19.2 Return on Investment From Open Source Contributions

The results of a 2020 study about perceived ROI (return on investment) from Open Source benefits are summarised in Figure 19.2.[1]

19.3 The Role of an OSPO, Model Options, and Where Should We Build It?

OSPOs often start with one individual who brings a vision of the scope of the company's Open Source vision and the OSPO and works hard to customise it for the company's unique business needs. Not all companies need the same type of OSPO. Each organisation should consider the options and create a model that works well for its individual culture and structure. In particular, an OSPO needs to help align Open Source engagement with the greater business strategy of the company. A centralised model works well in a company where a centre of excellence helps organise and create all policies and processes to manage Open Source work overseen by the company's engineering organisations. When there are many divisions, each doing vastly different product work, a decentralised model may work

[1] <https://github.com/todogroup/survey> accessed 18 April 2022.

best. In companies which have strong functional groups, like legal, marketing, and communications, a matrixed function may work. And in smaller organisations, a volunteer group or part-time OSPO may be most effective.

The scope of OSPOs also varies. Some focus only on compliance and work closely with the legal team and the development teams creating customer-facing products. They either run compliance tooling or build it and help to integrate it into development pipelines, as well as to help teams run scans and publish their disclosures. Often this type of OSPO is developer-heavy, because compliance tooling requires integration, customisation, and scripting to work.

Some OSPOs do a lot of marketing work. They put a specific focus on evangelising the company's Open Source work by sponsoring events, volunteering to speak at departmental meetings, etc. They leave the consumption, contribution, and compliance of Open Source components to the engineering teams to manage.

Increasingly, OSPOs handle multiple aspects of Open Source. The best way to characterise it is shown in Figure 19.3. Acting as a centre of excellence in all things Open Source, the OSPO guides the company in getting the highest return from the use and adoption of Open Source methodologies.

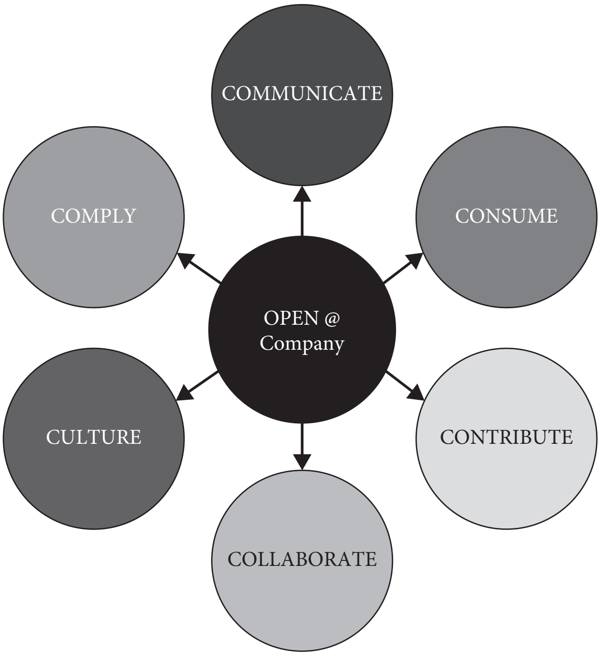


Figure 19.3 Various Functions of an OSPO

19.3.1 Drilling down into OSPO's components

Communication: An OSPO brings together development communities inside the company with consistent communications, guidance, and training on Open Source practices. It amplifies the company's Open Source work in the industry, deliberately to elevate the work and create positive relationships in the community. One often sees blogs, podcasts, talks, articles, and sponsorship from OSPOs at Open Source community events and an OSPO's staff gives back to the industry and community through sharing best practices and other contributions. Besides goodwill, this creates recognition for the organisation and eases the acceptance of their code; good communication and relationships also matter when an OSPO's staff needs community support.

Consumption: While it is easy for contemporary developers to download and use Open Source resources from the Internet, it is the OSPO, working with the legal team, that establishes smart and safe usage policies and practices. It is important to know and track the origin of downloaded Open Source code and to understand the health of the community from which resources were downloaded. Licences are also important. Proactive and consistent guidelines prevent hundreds of questions flooding into the legal department, by removing or mitigating confusion for new developers and creating embedded good practices.

Contribution: OSPOs coordinate reviews, guidance, and approval of code contributions from the company to Open Source projects. This is an important function as there are many elements to making high-quality contributions that will be successful. A company has to decide what to keep inside the company versus what to contribute, why contribution may be helpful, how to make a high-quality contribution and to host it in the right location.

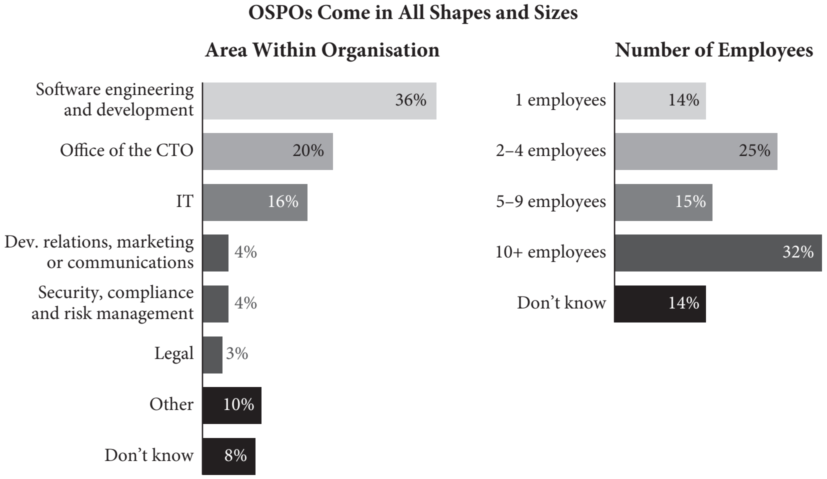
Collaboration: The very nature of Open Source work is that it is collaborative, often on a global scale. Hence, working with projects and foundations in areas of interest to the company are vital to learning and getting work done. It is also a great place to learn and share best practices.

Culture: Because engaging with Open Source reaches into many aspects of development in a company, it necessarily touches an organisation's culture on many levels. It changes the way the company sources and develops code, and it changes the relationship with upstream sources from vendors to Open Source communities. It also changes recruitment positively and other policies inside the company. An entire chapter could be written on the matter of Open Source, OSPOs, and corporate/company culture impacts alone.

Compliance: Because Open Source licences come with obligations, compliance is a critical function for any OSPO to take on. Most developers are busy developing, and as such may not pay attention to the licence or its obligations. It is the OSPO's responsibility to educate, set guidelines on what is acceptable, and provide guidance on how to follow the licence requirements. In this area, an OSPO works closely with any legal and corporate compliance functions to create policies and guidelines, and with engineering departments to ensure that risks are managed. An OSPO also answers any questions that arise on policies and licences.

The structure of an OSPO, where it lives, and the functions it performs are unique to each company's needs (see Figure 19.4). Most OSPOs 'live' in engineering, to guide developer relations and advocacy functions and to help manage developer education and requests. They serve as the first line of support before anything is escalated to legal, marketing, or leadership. OSPOs can also be housed in a CTO (Chief Technology Office) function, in marketing, or in legal. The best placement of an OSPO tends to be where it is closest to Open Source developers and users.

Large company OSPOs may have functions at the Director and VP (Vice President) levels, which closely coordinate with the CTO and/or Senior Vice President (SVP) of Engineering. These OSPO leaders work closely with leadership to provide an Open Source perspective and to stay aligned with business strategies. Smaller companies may appoint a single person at the Senior Manager level or Principal Engineer level. Because this function stands at the intersection of so



Source: "Open Source Programs in the Enterprise - 2020" Survey.
Left Chart: Where is the open source program or initiative located within the organisation? If the effort is informal, answer based on who the primary organizers report to. n = 275. Right Chart: How many employees are part of your open source program? n = 326.'

Figure 19.4 Where to locate your OSPO and Staffing an OSPO

many disciplines and impacts company risk, strategy, transformation, and even sustainability, it is a function that needs to ensure its strategy is aligned with company strategy.

19.4 How Did OSPOs Get Started and the What is the ToDo Group?

It was in the early 2000s that hyperscale technology companies like Google realised that it was beneficial to dedicate a small organisation to help their developers safely and smartly use Open Source and follow licences. Google's Chris Di Bona is credited with coining the term 'Open Source Program Office and started with a group of developers who were Open Source enthusiasts. Residing in engineering, they acted as the first line of contact for the hundreds of questions that were coming in from their developers on licences and other Open Source matters. It shielded legal teams from being overwhelmed by developers directly reaching out to them, usually with queries related to various licences. According to Chris Di Bono, who leads the Google OSPO, they saw it as vital way to enable developer productivity and excellence. That early OSPO also started taking on the function of managing Google's relationship with external development communities on behalf of the company. These were communities of projects that the company used and depended on. More information about Google's OSPO can be found at: <https://opensource.google/>.

In parallel, Intel created an organisation called 'OTC' (Open Source Technology Center) which handled upstream contributions, communicating and advocating for Open Source projects and helping make many projects enterprise-ready. One of the projects Intel championed and continues to Open Source is the Yocto Project, which is a standard for embedded Linux development today. Intel has historically been a prolific contributor to the kernel and a great supporter of many Open Source foundations. Intel's Open Source work can be found at: <https://01.org/>.

Other OSPOs in the early days include Box, Facebook, Twitter, Sun, and many other mainstream and non-mainstream technology brands. Comcast started working in Open Source in the early 2000s and in 2017 it established the Comcast OSPO because of its scale and strategy to lean-in to Open Source. Along the way, the people running the OSPOs often helped OSPOs in other organisations with questions and best practices. In 2014, a more formal organisation coalesced, called the ToDo Group, during Facebook's developer conference '@scale'. In 2016, the ToDo Group became a project in the Linux Foundation. The goal was to create a neutral place for company OSPOs to share, collaborate on common issues, and promote and support the growth of OSPOs. The ToDo group is one of the best places to learn about how to build and run an OSPO. Its members are generous with their knowledge and time because of their common goal to share and help others to make their OSPOs successful. Many case studies and other information can be found at: <https://todogroup.org/about/>.

The ToDo group defines an OSPO in this way:

> An Open Source Program Office (OSPO) is the center of gravity for an organisation's open source operations and structure. This can include training developers, ensuring legal compliance, engaging with and building communities, and defining policies that govern code usage, distribution, selection, auditing and more. <https://todogroup.org/blog/ospo-definition/>.

Today, large enterprises that use Open Source to speed their innovation typically establish an OSPO. Comcast's OSPO and contributions, for example, are housed at <https://comcast.github.io/>. CapitalOne's approach to Open Source can be found at <https://www.capitalone.com/tech/open-source/>.

Today, governments and universities are starting OSPOs. As such, the OSPO is a concept well understood for organising Open Source work aligned with the business and mission of an organisation. As an example of academic OSPOs, the Rochester Institute of Technology, or RIT, hosts an Open@RIT office; its mission can be found at: <https://www.rit.edu/news/rit-creates-openrit-university-wide-initiative-all-things-open>. Notably, the city of Paris is one of the pioneers in the use of Open Source and the establishment of an OSPO to help promote and manage its use; here is a link to more information about that effort: <https://www.smartcitiesworld.net/special-reports/special-reports/paris-uses-open-source-to-get-closer-to-the-citizen>. As well, here is information on the UK government's approach to Open Source: <https://gds.blog.gov.uk/about>.

19.5 What is the Impact of an OSPO on an Organisation?

It is easy to claim that Open Source adoption and use happens organically in an organisation, which leads to dangerous oversimplifications and assumptions along the lines of 'why can't people who have questions just go to legal when they need to contribute?' and 'does an OSPO really make a difference?' The answer is yes, it does. Especially in a large or complex organisation, the OSPO can bring a great deal of order and strategic thinking to Open Source engagement.

Legal and Efficient Use of Open Source Code: For most organisational legal teams, Open Source activities represent a fraction of their overall responsibilities. However, there are various elements of Open Source engagement that require legal guidance, especially as it relates to licences. This is where the OSPO can be a highly valuable first line of support. OSPOs work hand-in-hand with distinct parts of legal departments to create guidelines, policies, and processes that reduce friction and create more productivity for developers using and engaging with Open Source.

Because it is so easy to download Open Source projects and fragments without understanding the licence, the matter of community health and security vulnerabilities can introduce a number of risks. OSPOs establish education and guidance to make consumption efficient and compliant with policy. An OSPO routinely fields hundreds of questions about existing and new licences and new software that developers want to use.

Following licence obligations when shipping Open Source products outside the organisation is another area of risk that is best handled by an OSPO. Licence obligations typically become activated upon shipment, which means that development teams need to build the discovery and documentation of their Open Source bill of materials when releasing a product. OSPOs work with legal colleagues to create policies around what the company believes can and cannot be used, how to capture it, and how to present it in the product in a way that standardises and simplifies compliance. The productivity hit that an organisation can take is high when there is no clarity on what to do or where to go to with questions.

An OSPO gets involved wherever due diligence is needed, to ensure that Open Source is being used effectively, and that policies and licence obligations are being met. Other areas where OSPOs can lend insights and credence is in M&A (Mergers & Acquisitions), related due diligence, and any investment-related research organisations make.

OSPOs help organisations make decisions about intellectual property, in terms of when to open resources versus keeping them inside the company. As such, OSPOs create governance and provide facilitation on code contributions that development teams want to make to the community at large. They ensure that all the right organisations across the company can weigh in and do the due diligence to make the contribution and the developer successful. Many steps go into reviewing and ensuring that Open Source community contributions are secure and of high quality. Once Open Source items are released, OSPOs help the maintainers and developers do the right thing, in terms of community growth and maintaining a healthy community.

Understanding Company's Use: There's a lot of value in OSPOs helping drive developer efficiencies by cataloguing all the Open Source being used and guiding engineering organisations on duplicative use and dependencies. A canonical case of this would be seeing that an organisation is using Vue, React, and Angular across all organisations, is that a good thing for the organisation? Should they standardise on one to improve onboarding and engineering efficiency? And by understanding dependencies, a company can manage which communities it works with and how it responds to security alerts as it knows what is being used and where.

Collaborative Development and Reuse: Because Open Source communities have existed for decades, they provide particularly good development practices from which to learn about collaborative development best practices. These can be brought inside the company to improve both the velocity and quality of development. 'InnerSource' is a term used to define bringing Open Source projects inside the firewall. Such elements include a more componentised architecture, better documentation, and better governance for others to contribute to the project, to name a few. This is a relatively new application of Open Source methodologies that OSPOs lead inside a company today and represents a big growth area for OSPOs. For a long time, companies have tried to break down development silos, leverage talent across the company, to reuse work and not reinvent. InnerSource does just that. At the most fundamental level, it improves development and engineering practices, by ensuring that even the largest and most diverse organisations can benefit from the best software development efforts from across their various divisions and groups. You can check out more information on InnerSource practices at <http://.www.Innersourcecommons.org>.

Collective Innovation: According to a ZDNet article,[2] over 78 per cent of firms use Open Source to run their companies. It is hard to avoid Open Source as most proprietary products and cloud services all use Open Source innovation in one form or the other. Clearly if it is in use, making competent use of Open Source can be a competitive advantage to a company. Strategic questions about where to Open Source, when the company should keep inside the company, when to release a project, which communities with which to collaborate, and how to manage communities around company projects can create competitive advantages. It can also help companies get to market in a way that is 'faster, better, cheaper'. Open Source is an acknowledged new way to create *de facto* standards in a certain technology area. A solid Open Source reputation often helps attract developers to the company, as well as helping retain developers inside a company. Managing working with external communities, foundations, and other companies in an efficient way is a key part of an OSPO's role, external to the company.

Measuring Value and Total Cost of Ownership: While in the early days of Open Source, people consumed it because it was free, and we do still see a rise in adoption in times of financial downturn,[3] cost is usually much lower on the list of reasons why people consume Open Source today. The cost advantages, however, cannot be ignored, especially in large and publicly held companies, focused on demonstrable and quarter-to-quarter financial returns. Imagine having to create every bit of the software stack, from scratch, that is commonly used in a product or service. This slide below from the Linux Foundation shows that 'best-in-class' companies use 80 per cent Open Source and 20 per cent proprietary software in their product code. That would take years and cost as much as or more than four times today's development costs if one had to develop all the components in a stack. And as a direct result, time-to-market would suffer. A small team inside a company can do so much

[2] <https://www.zdnet.com/article/its-an-open-source-world-78-percent-of-companies-run-open-source-software/> accessed 18 April 2022.

[3] TideLift Report.

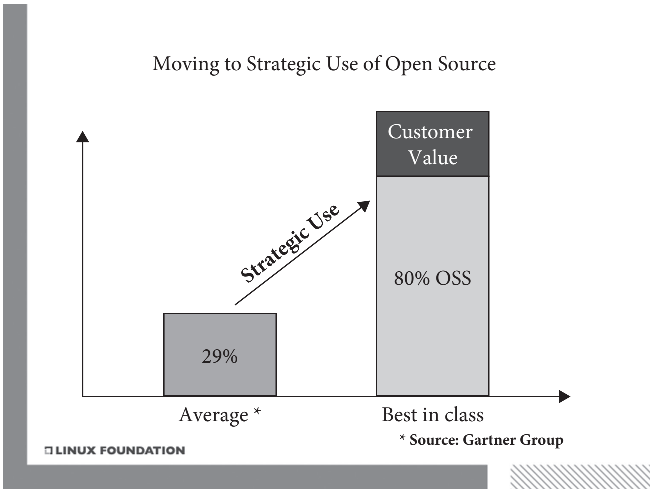
more by using the collective innovation of a global community. Besides the use, companies can access source code and customise it for their use and, by contributing back, thereby reduce the technical debt of maintaining software. So, the cost efficiencies associated with using Open Source certainly cannot be ignored.

Total cost of ownership was a convenient economic model when considering on premises use for Open Source software and has gained acceptance as it is popular with proprietary companies in a non-platform environment. It has been ideal to demonstrate that the costs of software utilisation extend beyond royalty or licence fees and to demonstrate that Open Source is not free of cost. However, economic models are discussed in more depth in Chapters 15 and 16.

19.6 How to Get Started in Creating Your Own OSPO?

If you are a serious user of Open Source and want to move with more intention and strategically, starting an OSPO is an important first step (see Figure 19.5). Some of the key steps to starting your own OSPO include:

1. Find a leader for this office who understands deeply how Open Source works and can bridge the organisation's business with Open Source strategy. Think about selecting someone whose full-time job is to think about Open Source strategy for the company.
2. Make sure that the Open Source strategy is well understood and supported by your organisation's technical and business leadership, including middle



* Source: Gartner Group

Figure 19.5 Best in Class Use of Open Source

managers and developers. Without the support of leadership ‘up and down the chain’, an OSPO becomes merely tactical, and the highest strategic benefits are not fully realisable.

3. Create an operational model that works for your specific organisation. Whether centralised or decentralised, establish clear guidelines, processes, and responsibilities across legal, development, and OSPO, regarding who does what.
4. Work with and join foundations and communities that matter to the company, based on company dependencies. Some of the most important organisations in Open Source today are the Linux Foundation, where hundreds of projects are housed; the Apache Foundation, where additional hundreds of key community-run projects are housed; and the Eclipse Foundation. There are many other key institutions in Open Source, like the Open Source Initiative (OSI), that review and approve new licences as aligned with the Open Source Definition.
5. Work with other OSPOs in the ToDogroup.org, to learn, share, and collaborate on familiar and unfamiliar challenges.
6. Be visible in the community. Elevate and support the community to thrive and encourage contributions back to Open Source. Without all companies contributing back code, time and money, the Open Source commons or collective innovation from which we all benefit will not exist.

19.7 Conclusion and Attributions

Software is at the heart of organisations, accelerating their digital strategy and transforming how their stakeholders work with the organisation. And most of today’s software is created using Open Source tools, libraries, services, and methodologies. Open data and open standards are other tools increasingly important to a company’s digital strategy. Managing this vital supply chain in a proactive and systematic way is business-critical for an organisation. OSPOs help companies understand, organise, and align Open Source strategies to the organisation or company’s business goals and strategies.

20

Cloud Native Development, Containers, and Open Source Licensing

Richard Fontana

20.1 Overview of Linux Containers

A container is a Linux operating system feature that enables isolation of a user-space instance from the rest of the system. Containers provide a lightweight form of virtualisation in comparison to the hardware virtualisation enabled by the use of hypervisor technology.[1] The Docker project popularised an easy way of building and sharing containerised applications.[2] Under this approach, a container image format, featuring a set of immutable filesystem layers, is used for packaging and distributing software intended to be run in containers. Container images are delivered through a network service known as a registry. A container image may store hundreds of applications, utilities, and libraries, and typically includes a 'base layer' consisting of a stripped-down Linux distribution without the kernel. Various aspects of container technology are now undergoing standardisation by the Open Container Initiative.[3]

[1] 'OS-level virtualization', <https://en.wikipedia.org/wiki/OS-level_virtualization> accessed 2 August 2020.

[2] The Docker community project, launched in 2013, was partially rebranded as 'Moby' in 2017. See 'Moby Project', <https://github.com/moby/moby> accessed 2 August 2020. There are now alternative Open Source implementations of Docker-style container technology, such as the buildah, podman, and skopeo projects maintained by Red Hat. See 'Say "Hello" to Buildah, Podman, and Skopeo', <https://servicesblog.redhat.com/2019/10/09/say-hello-to-buildah-podman-and-skopeo/> accessed 2 August 2020.

[3] 'Open Container Initiative', <https://opencontainers.org/> accessed 2 August 2020.

Richard Fontana, *Cloud Native Development, Containers, and Open Source Licensing* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Richard Fontana 2022. DOI: 10.1093/oso/9780198862345.003.0020

Enterprise cloud computing has become closely associated with a so-called cloud native approach to building and running scalable applications in public, private, and hybrid clouds. Cloud native development centres around the use of container technology along with the adoption of DevOps practices. Cloud native applications are structured as a set of containerised microservices, each of which focuses on performing one service.[4] Container orchestration tools, the most widely adopted of which has been the Kubernetes project, are used to dynamically coordinate the multiple containers making up an application.[5]

Containers typically include a large amount of Open Source software, for the most part ultimately derived from upstream community projects, even in cases where the container is focused on providing a proprietary service. This is so for two overlapping reasons. First, much of the software included in a container image consists of packages maintained by the underlying Linux distribution the container is based on (including the packages in the base layer). Second, containerised applications are no different from contemporary non-containerised software in that application runtime environments consist largely of Open Source-licensed dependencies. The licence makeup of the open source part of a container application runtime invariably includes both copyleft and permissive (non-copyleft) licences. Copyleft licences in the General Public Licence (GPL) family, primarily the GNU GPL version 2 (GPLv2),[6] GNU GPL version 3 (GPLv3),[7] and GNU Library or Lesser GPL versions 2.0 and 2.1 (LGPLv2.0[8] and LGPLv2.1[9]), are particularly prevalent in the subset of the container runtime that consists of Linux distribution packages. This reflects the licence composition of the most fundamental user-space packages in mainstream Linux server distributions, which are largely GPL-licensed.[10]

Given the abundance of Open Source software in containers, issues of Open Source licence interpretation and compliance naturally arise in the container setting. One of the characteristics of the rapid rise of container technology adoption has been a perceived inattention to Open Source licence compliance by many

[4] 'CNCF Cloud Native Definition v1.0', <https://github.com/cncf/toc/blob/master/DEFINITION.md> accessed 2 August 2020.

[5] 'Kubernetes', <https://kubernetes.io/> accessed 2 August 2020.

[6] <https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html> accessed 18 April 2022.

[7] <https://www.gnu.org/licenses/gpl-3.0.html> accessed 18 April 2022.

[8] <https://www.gnu.org/licenses/old-licenses/lgpl-2.0.en.html> accessed 18 April 2022.

[9] <https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html> accessed 18 April 2022.

[10] See, e.g., Red Hat Universal Base Image 8, <https://catalog.redhat.com/software/containers/ubi8/ubi/5c359854d70cc534b3a3784e?container-tabs=packages> accessed 18 April 2022. For a good historically oriented discussion of containers, see Daniel Riek, 'A Greybeard's Worst Nightmare: How Kubernetes and Containers are re-defining the Linux OS', <https://www.redhat.com/files/summit/session-assets/2017/S104999-graybeard-rhsummit2017.pdf> accessed 18 April 2022.

container users.[11] To the extent this is a fair assessment, it may reflect the ease with which a user of container technology can assemble and distribute complex, Open Source-based software collections without awareness of their contents or associated licensing terms. At the same time, because containers may be new and unfamiliar to lawyers, they may be overly concerned about the impact of Open Source licence obligations arising out of the use of containers.[12]

20.2 Containers and the Scope of Copyleft

The 'scope of copyleft' issue in Open Source licensing centres on interpretation of the GPL (see an overview of licensing and copyright at Chapter 3). With the increasing adoption and awareness of containers, some have raised concerns that in the container setting the GPL's copyleft is broader in scope than it would be in counterpart non-containerised scenarios.[13] This section provides some background on the issue and explains that containers do not warrant any alteration of the general interpretation of GPL copyleft.

Three broad technical observations may be useful for understanding the GPL copyleft scope issue. First, software is typically developed through the composition of smaller, modular components. The additional components needed by an application at runtime are termed the runtime dependencies of the application. With the rise of Open Source, a growing portion of an application's runtime dependencies are drawn from Open Source project codebases, typically as packaged by a major Linux distribution or in a standard package repository for a particular programing language community.[14] This characteristic of modern software development is particularly evident in containers, since containers isolate an application along with its user-space stack.

Second, an application's dependencies may be divided into two categories, library dependencies and system dependencies,[15] though the distinction is not

[11] See Jake Edge, 'An update on compliance in containers' (16 April 2019), <https://lwn.net/Articles/786066/> accessed 18 April 2022; Armijn Hemel, 'Docker containers for legal professionals', <https://www.linuxfoundation.org/publications/2020/04/docker-containers-for-legal-professionals/> accessed 18 April 2022.

[12] One common use case for containers involves mere internal consumption, which should not justify concerns about Open Source licence compliance, given that, for all practical purposes, the obligations of Open Source licences are triggered by distribution.

[13] See Richard Fontana, 'Containers, the GPL, and copyleft: No reason for concern', 24 January 2018, <https://opensource.com/article/18/1/containers-gpl-and-copyleft>; Dirk Riehle, 'The GNU Public License v2 in the land of microservices', 17 June 2020, <https://dirkriehle.com/2020/06/17/the-gnu-public-license-v2-in-the-land-of-microservices/> both accessed 8 August 2020.

[14] Some important examples of the latter type of package repository are Maven Central, <https://repo1.maven.org/maven2/> (Java); PyPI, <https://pypi.org/> (Python); and npm, <https://www.npmjs.com/> (Node.js), all accessed 18 April 2022.

[15] This terminology is not widely used but captures a useful distinction. It was suggested to the author by Van Lindberg in a mailing list discussion.

always clear-cut. Libraries are collections of modular, reusable software functions that are called by a program to access their functionality.[16] The libraries used by an application are typically written in the same programing language as the application, and the application and the libraries it invokes will typically run in the same operating system process (i.e. an executing instance of a program). An application's system dependencies will generally be run in separate processes and often involve a different programing language runtime.

Third, as a consequence of various technical and commercial developments in programing languages and operating systems over the past few decades, software development can be said to have become increasingly 'loosely coupled' in style. The earliest GPL-licensed programs in the late 1980s were typically written in C, a relatively low-level systems language, and compiled programs typically consisted of a single executable object code file including all libraries statically linked into the application.

Today, lower-level user-space programs (those that interact more directly with real or virtual hardware) more typically use dynamically linked libraries shared among processes, and enterprise application development has increasingly favoured higher-level programing languages (such as Java, JavaScript, and Python), which rely on a virtual machine or interpreter and dynamic loading of library modules.[17]

The GPL has a number of interrelated requirements associated with the term *copyleft*, which may be summarised as follows:

1. Derivative works of a GPL-licensed work, if distributed, must be licensed 'as a whole' under the GPL.[18]
2. Binary versions of a GPL-licensed work, if distributed, must be accompanied by complete corresponding source code in one of a number of specified ways.[19]

[16] For a good judicial discussion of the related topic of APIs, see *Oracle America, Inc. v Google, Inc.*, 872 F. Supp. 2d 974 (N.D. Cal. 2012), *rev'd*, 750 F.3d 1339 (Fed. Cir. 2014), *cert. denied*, 135 S. Ct. 2887 (2015).

[17] See John R Levine, *Linkers and Loaders* (San Francisco: Morgan Kaufmann Publishers 2000); see also Mark Radcliffe, 'Top 10 FOSS issues of 2012', 11 January 2013, <https://opensource.com/law/13/1/top-ten-foss-2012> accessed 8 August 2020, (noting 'the rise of "loosely coupled" programming techniques').

[18] GPLv2 § 2; GPLv3 § 5. Note that, unlike GPLv2, GPLv3 does not use the US copyright statutory term 'derivative work', instead relying on a definition of 'modified version' that references permissions under background copyright law. GPLv3 § 0 ('To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a 'modified version' of the earlier work or a work "based on" the earlier work.'). Despite the change in terminology in GPLv3, informal discussions of GPL copyleft interpretation have generally continued to refer to derivative works even in contexts where GPLv3 is the relevant licence. GPLv3 is not intended to expand or narrow the general scope of copyleft as it was interpreted under GPLv2, though in certain specific ways it relaxes the strictures of GPLv2 copyleft (particularly with respect to compatibility of other licences with the GPL).

[19] GPLv2 § 3; GPLv3 § 6. Compliance with this requirement in the container setting is discussed in the following section.

3. Distributors may not impose ‘further restrictions’ on recipients’ exercise of permissions under the GPL.[20]

As to the derivative work requirement, the copyleft provisions of GPLv2 more precisely use the term ‘work based on the Program’, which in turn is defined by reference to derivative works. The term ‘the Program’ refers to the work originally licensed under the GPL. GPLv2 defines ‘work based on the Program’ as ‘either the Program or any derivative work under copyright law; that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language’.[21] Some lawyers have seen this as a sweeping partial redefinition of the statutory term ‘derivative work’ (see also Chapter 3). However, GPLv2 also has a ‘mere aggregation’ clause that makes clear that the scope of copyleft, and in particular the breadth of what can be considered a ‘work based on the Program’, is limited: ‘[M]ere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.’[22]

From the earliest years of the GPL in the late 1980s, there has been a debate in the technical community over the interpretation of ‘derivative work’ as used in GPLv2. A preoccupation with the derivative work issue later developed among lawyers as they became more familiar with Open Source licensing. A thorough exploration of this topic is beyond the scope of this chapter but is dealt with in Chapter 3. It is sufficient to note that there is a long-standing view, widely held among GPL licensors and Open Source community members generally, that under the GPL derivative works encompass notional combinations of a GPL-licensed work with other software having a library dependency relationship with the GPL-licensed software.

The view that a derivative work of GPL-licensed software extends across (library) dependency boundaries in at least some circumstances is largely the result of the influence of the Free Software Foundation (FSF), the drafter and

[20] GPLv2 § 6; GPLv3 § 10.

[21] GPLv2 § 0. See also GPLv2, final paragraph (‘This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.’).

[22] GPLv2 § 2, last para. The more complex counterpart provision of GPLv3 states:

> ‘A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.’

GPLv3 § 5, last para. Cf. OSD para 9, <https://opensource.org/osd/> (‘The licence must not place restrictions on other software that is distributed along with the licensed software’) accessed 21 July 2022.

maintainer of the GPL, on community interpretation of the GPL, particularly given the general absence of case law interpreting the licence as well as the lack of helpful case law on the meaning of 'derivative work' in the software context.[23] It is likely that the FSF's views on the policy issue—how far should the GPL copyleft requirements extend to downstream combinations of GPL and non-GPL components—were influenced by the shift towards more 'loosely coupled' programing following the release of GPLv2 in 1991. In some of its interpretive materials, the FSF has associated derivative work scope with the notion of a 'single program'. Library dependencies, under this view, were historically statically linked into the same executable file as the rest of the program, and the GPL should not be interpreted to have a weaker copyleft scope in a 'loosely coupled' counterpart program.[24]

On the other hand, the FSF-influenced copyleft interpretation would generally not view system dependencies as falling within GPL copyleft scope. Rather, an application and its system dependencies would be 'mere aggregation'. The FSF provides a technical rationale:

> Where's the line between two separate programs, and one program with two parts? This is a legal question, which ultimately judges will decide. We believe that a proper criterion depends both on the mechanism of communication (exec, pipes, rpc, function calls within a shared address space, etc.) and the semantics of the communication (what kinds of information are interchanged).
>
> If the modules are included in the same executable file, they are definitely combined in one program. If modules are designed to run linked together in a shared address space, that almost surely means combining them into one program.
>
> By contrast, pipes, sockets and command-line arguments are communication mechanisms normally used between two separate programs. So when they are used for communication, the modules normally are separate programs. But if the semantics of the communication are intimate enough, exchanging complex internal data structures, that too could be a basis to consider the two parts as combined into a larger program.[25]

Using examples familiar to technical Unix and Linux users, this explanation relies on the significance of the process boundary in distinguishing 'mere aggregation' from derivative work.

[23] See Dan Ravicher, 'Software Derivative Work: A Jurisdiction Dependent Determination', 13 November 2002, <https://www.linux.com/news/software-derivative-work-jurisdiction-dependent-determination/> accessed 8 August 2020.

[24] Cf. GPLv3 § 5, last para (definition of 'Aggregate' suggests that a work based on the Program does not include a compilation of works 'which are not combined . . . such as to form a larger program').

[25] 'Frequently Asked Questions about the GNU Licenses', <https://www.gnu.org/licenses/gpl-faq.en.html> accessed 9 August 2020.

The limited scope of GPL copyleft is consistent with long-established practice and assumptions among distributors and users of traditional (non-containerised) Linux. For example, there has never been a serious contention that the GPL extends to the entirety of Linux user-space merely because many user-space packages are licensed under the GPL. Indeed, many packages in Linux distributions are under GPL-incompatible licences. Moreover, proprietary software is commonly distributed for execution on, along with, or as part of Linux distribution products. Linux-based virtual machine images, which are somewhat analogous to container images, often include proprietary software in addition to GPL- licensed and GPL-incompatible Open Source software.

We can now address the specific topic of copyleft scope in the container context. There is anecdotal evidence of a belief that all the code in a container, and perhaps even all the code in a multi-container implementation of an application, necessarily becomes subject to the GPL's copyleft requirements because the container causes it all to be a derivative work of one or more of the container's GPL-licensed components. Such views are likely to reflect a fear of unfamiliar technology, along with a misimpression caused by the use of the term 'container'. To those not technically knowledgeable about containers, the term might seem to suggest a container is analogous to a modular package or a 'single program', to use the FSF's phrasing. However, containers are so called because of the isolation they achieve, and not because they 'contain' anything.

A company concerned about the potential effects of distributing GPL-licensed code along with its own proprietary software might attempt to prohibit its developers from adding any third-party GPL code to a container image it plans to push to a registry or distribute by way of a registry. To the extent the goal is to avoid distributing GPL-licensed code, this is a dubious strategy. As noted earlier, the base layer of any normal container image will include many GPLv2- and GPLv3-licensed packages. While the company might be able to prevent its developers from including GPL-licensed software in the container image layers they create, it generally cannot guarantee that when it pushes a container image to a registry it will never push the base layer or other GPL-code-containing third-party-created layers that its container image is building on.

In any case, the concern about having both proprietary software and GPL-licensed software running in the same container, or distributed in the same container image, is misplaced, because the two components are no more likely to have a copyleft-significant relationship in the containerised case than in the well-understood non-containerised case. In general, unless there are particular facts suggesting that the two components are not 'separate and independent', the simultaneous inclusion of a proprietary component and a GPL-licensed component in a container image will be 'mere aggregation', compliant with and contemplated by the GPL, with no impact on the licensing of either component. While in a given situation the relationship between two components may not be 'mere aggregation', the same is true of software running in non-containerised user-space on Linux, or any other mainstream operating system. That is, there is nothing in the technical makeup of containers or container images that suggests a need to apply a different, expanded form of copyleft scope analysis.

It follows that when examining the relationship between code running in a container and code running on the same Linux system outside a container, the 'separate and independent' criterion suggested by the GPL text and FSF interpretive guidance is almost certainly met. The two components in this case will necessarily run as separate processes, and the whole technical point of using containers is isolation from other software running on the system.

A more practical scenario would involve separate but potentially interacting containers, as part of a containerised application with a microservices architecture. Each container will undoubtedly have GPLv2 and GPLv3-licensed software, in addition to code under various other Open Source licences. Suppose one of the interacting containers also has some proprietary code. In the absence of very unusual facts, copyleft scope should be expected not to extend across multiple containers. Separate containers run in separate processes. Communication between containers by way of network interfaces is analogous to the operating system mechanisms cited by the FSF as examples of mere aggregation (e.g. pipes and sockets). Moreover, a multi-container microservices scenario would seem to preclude what the FSF calls 'intimate' communication, by definition. While the composition of an application using multiple containers may not be dispositive of the GPL scope issue, it makes the technical boundaries between the components more apparent and provides a strong basis for arguing separateness. Here too there is no technical characteristic of containers or container images that suggests application of a different and stricter approach to copyleft scope analysis.

The preceding discussion suggests that a company concerned about the impact of GPL code distribution on simultaneously distributed proprietary code might wish to embrace containerisation as a strategy for minimising copyleft scope concerns, by isolating GPL and proprietary code from one another. This may not always be practical (e.g. if the GPL code is a system dependency of the proprietary code, it may be necessary for it to run in the same container). It would be more appropriate for such basic architectural decisions to be driven by technical considerations, rather than often-unfounded or overblown legal concerns about the impact of the GPL that fail to account for mere aggregation. While in a non-containerised setting the relationship between two interacting components will often be mere aggregation, the evidence of separateness that containers provide may be comforting to those who worry (however needlessly) about GPL copyleft scope.

20.3 Container Images and Source Code Compliance

Licences classified as copyleft typically impose some requirement on distributors of binaries to provide source code. As noted earlier, this is true of the GPL with its relatively elaborate 'complete corresponding source code' requirement.[26] The source code disclosure requirements in other, less commonly used copyleft licences are generally worded in a less detailed manner and are often assumed to be generally satisfied through attention to GPL-style source compliance, although it is not clear that this is entirely correct.[27] In contrast to source code requirements, nearly all Open Source licences include requirements of various sorts to preserve legal notices such as copyright statements, licence notices, licence texts, and in some cases author attributions.[28]

Making source code available simultaneously with distribution of binaries is an efficient way of complying with both types of basic Open Source licence requirements, since source code will generally contain all legal notices for which licences mandate preservation, though this does not appear to be widely recognised by vendors engaged in Open Source licence compliance efforts.[29] Distribution of container images, for example distribution by way of a registry, will, of course, give rise to all the distribution-triggered Open Source licence requirements that would be triggered in a non-containerised setting.

GPLv2 section 3 gives two options for satisfying the corresponding source code requirement in the case of commercial distribution of object code:[30]

[26] GPLv2 defines complete corresponding source code: 'For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable.' GPLv2 § 3. GPLv3 defines the term 'Corresponding Source':

> 'The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities .... For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.' GPLv3 § 1.

[27] See, e.g., Eclipse Public Licence 2.0 § 3.1 (if distributed in object code, 'the Program must also be made available as Source Code ..., and the Contributor must accompany the Program with a statement that the Source Code for the Program is available under this Agreement, and informs Recipients how to obtain it in a reasonable manner on or through a medium customarily used for software exchange'); Mozilla Public Licence 2.0 § 3.2 ('If You distribute Covered Software in Executable Form then ... such Covered Software must also be made available in Source Code Form ..., and You must inform recipients of the Executable Form how they can obtain a copy of such Source Code Form by reasonable means in a timely manner, at a charge no more than the cost of distribution to the recipient').

[28] See, e.g., Apache Licence 2.0 § 4.

[29] Jeffrey Robert Kaufman, 'An economically efficient model for Open Source software license compliance', 1 September 2017, <https://opensource.com/article/17/9/economically-efficient-model>; Heather Meeker, 'Is the container half empty or half full?', <https://heathermeeker.com/2020/07/29/is-the-container-half-empty-or-half-full/> both accessed 9 August 2020.

[30] For non-commercial distribution, GPLv2 also allows the distributor to '[a]ccompany [the object code] with the information you received as to the offer to distribute corresponding source code'.

> a) Accompany [the object code] with the complete corresponding machine-readable source code, which must be distributed … on a medium customarily used for software interchange; or,
>
> b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed … on a medium customarily used for software interchange ….

In addition, the last paragraph of section 3 states:

> If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

Though it is not immediately obvious from the licence text, this is generally understood as a clarification that the first option (simultaneous source availability) is available when object code is distributed over a computer network, which of course today is the normal manner in which object code is commercially distributed. It should be remembered that when this language was added to GPLv2 in 1991, it was still common for free software to be distributed using physical tape media, and large-scale network distribution of software was still in its infancy.[31]

The source availability options in GPLv3 for commercial distribution of object code are similar, though with some notable changes:

> a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
>
> b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost

GPLv2 § 3(c). GPLv3 has substantially the same option but limits it to being used only 'occasionally'. GPLv3 § 6(c).

[31] Cf. GPLv1 (1989) § 3 (containing no counterpart to the 'equivalent access' clause of GPLv2).

> of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
>
> …
>
> d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
>
> e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.[32]

Under GPLv3 the written offer option appears no longer to be available when distribution ('conveying') of object code is not done by way of a hardware device or physical storage medium. The simultaneous-source option, which now explicitly includes the network binary distribution case, is liberalised somewhat to codify FSF interpretation of GPLv2.[33]

The written source offer option is a possible form of source compliance for network binary distribution under GPLv2, and is one commonly chosen by vendors in that context, perhaps because it can seemingly provide a quick solution for last-minute GPL compliance problems. However, written offers have a number of drawbacks.[34]

A benefit of the simultaneous source availability approach is that compliance is fully achieved at the time of object code distribution, whereas with the written offer option there is an ongoing and somewhat unpredictable source compliance obligation based on the required term of the offer. Moreover, in addition to its narrowed availability under GPLv3, it is not clear that the written offer approach would comply with the source availability requirements of some non-GPL copyleft licences, nor would it seem to take care of notice preservation compliance adequately under Open Source licences generally. For these reasons, the written offer

[32] GPLv3 § 6.

[33] See 'Frequently Asked Questions about version 2 of the GNU GPL', <https://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.en.html> accessed 9 August 2020.

[34] See Eben Moglen and Mishi Choudhary, *Software Freedom Law Center Guide to GPL Compliance*, 2nd edn, (2014) 52–54, available at <http://softwarefreedom.org/resources/2014/SFLC-Guide_to_GPL_Compliance_2d_ed.pdf> accessed 9 August 2020; *Copyleft and the GNU General Public License: A Comprehensive Tutorial and Guide*, part II, § 15.1.2, available at <https://copyleft.org/guide/comprehensive-gpl-guidech16.html#x21-13100015> accessed 9 August 2020.

option should be avoided in the network binary distribution context, including distribution of container images, other than perhaps as a 'belt and braces' mechanism. An additional problem with the written offer approach in the container setting is that there is no straightforward way to present the offer to the recipient of the container image.

Because containers invariably include a large amount of Open Source-licensed software, distribution of container images gives rise to the same source availability requirements that would be triggered in a corresponding non-containerised distribution setting. However, in some important respects container images differ from non-containerised software, in ways that may make Open Source licence compliance more challenging.

In the past, prior to the rise of containers, modular software components were commonly obtained separately. For example, a Linux user seeking to install some additional packaged software (corresponding generally to a particular upstream Open Source project) would install just that package along with any missing dependencies; the resulting distribution event would typically involve a relatively small number of packages. In contrast, even though containers are often focused on a particular application or service, a container image will include a heterogeneous collection of software—often hundreds of components, more analogous to the installation of an entire (non-containerised) working Linux distribution. This represents a shift in the focus of software delivery from one driven by how the software is built to one driven by how the software is used.[35] A typical creator or consumer of a container image will not be aware of the scale and complexity of the contents of the image, including with respect to licensing.[36]

In the shift to containers, one compliance-related challenge concerns the diminished role of package management. Prior to the rise of containers, package maintainers and package management tools played a key role in facilitating source availability and thus Open Source licence compliance. The focused nature of a package, the role of a package maintainer in Linux distributions and upstream language-specific package repositories, and the tooling that has been built to support package management systems has resulted in an expectation that the package maintainer will take responsibility for ensuring that source code is available. In Linux distributions, at least, tools that build binaries also collect the corresponding source code into an archive that can be delivered along with the binaries. The result is that most users of these systems have not needed to be concerned with source

[35] Scott K Peterson, 'Making compliance scalable in a container world' (9 July 2020),available at <https://opensource.com/article/20/7/compliance-containers>; Scott K Peterson, 'What's in a container image: Meeting the legal challenges' (31 July 2018), available at <https://opensource.com/article/18/7/whats-container-image-meeting-legal-challenges> both accessed 9 August 2020.

[36] See Jake Edge, 'Containers and license compliance' (2 May 2018), available at <https://lwn.net/Articles/752982/> accessed 9 August 2020.

availability. Source code is routinely available in the same format as the delivery of the executable software, using the same distribution mechanism.

For example, many Linux distributions use the RPM Package Manager (RPM) package management technology, in which binary packages are delivered in RPM format, and corresponding source code is normally available in a corresponding source RPM. In contrast, there is no convention for making available the source code corresponding to the executable software in a container image.

Another compliance-related challenge associated with the adoption of container technology is an increase in the number of distributors of Open Source software inexperienced with the requirements of Open Source licences. Companies that are beginning to offer their proprietary applications as container images may be facing GPL compliance obligations for the first time (see Chapter 5).

Determining what the corresponding source code is for a set of binaries is more challenging for container images than it is for conventional modular packages. With traditional server and desktop-oriented Linux distributions, source code is normally available to the person building the package. In contrast, binaries in container images are typically built using components previously compiled by third parties other than the container creator. When the container image is built, the source code for those components may or may not be readily at hand, depending on how the binaries were acquired and how they were built.

Recently Scott Peterson of Red Hat has described a way of solving these challenges by implementing a 'registry-native' approach to source code availability: delivering source code itself as a container image, through container registries. An Open Container Initiative (OCI)-conformant container image includes an image manifest which identifies the other elements of the image, including the various filesystem layers (each of which is a tar archive) making up the image. In the registry-native approach, a list of corresponding source artifacts is arranged as an image manifest, enabling the registry to serve those source artifacts in the same way it serves other container image parts. The layers of the source image are those source artifacts. Tools for moving container images can be directly applied to move the source artifacts.

There are a number of compliance-related benefits to adoption of the registry-native approach. It clearly satisfies the 'equivalent access' criterion of the GPL.[37] Compliance is addressed in an efficient manner, at the time of creation of a container image, avoiding the need to maintain additional, external processes and

[37] Patrick McHardy, a former Netfilter contributor who has gained notoriety through a series of controversial GPL compliance litigation actions in Germany, has reportedly raised GPL non-compliance claims based on failure to provide equivalent access. Greg Kroah-Hartman, 'Linux Kernel Community Enforcement Statement' (16 October 2017), available at <http://kroah.com/log/blog/2017/10/16/linux-kernel-community-enforcement-statement/> accessed 9 August 2020. While those claims may be without foundation, they have had the side effect of calling general attention to compliance with the equivalent access requirement.

mechanisms for making source code available. Compliance is also made portable: when an image is moved from one registry to another, it remains in compliance. The same tooling that moves executable container images can be used to move, store, and serve the source images on all hosting registries. Finally, the registry-native approach can take advantage of the deduplication capability that is inherent in the layered container image format, by storing separate source artifacts for each of the hundred or more software components from which the container image is built.[38]

20.4 Identifying the Licence of a Container

In recent years the term 'Open Source compliance' has come to be applied to activities that do not focus on substantive compliance with provisions of Open Source licences, but which instead centre upon compiling lists of information about the Open Source contents of software products.[39] The focus is on the discovery of what Open Source-derived software is in a product and how such software is licensed, as well as on how best to maintain, present, and share that information (see Chapter 6 on OpenChain and Chapter 7 on SPDX).

This sort of 'compliance' orientation can be expected to be increasingly directed towards containers as adoption of containers continues to grow. It has already occasionally manifested in discussions and activities in the community and among companies relating to how to describe or identify the 'licence' of a container. These activities sometimes reveal a failure to appreciate the licensing complexity of container images, even among those having technical familiarity with containers.

For example, in the OCI Image Format Specification,[40] one of the predefined annotation keys is 'org.opencontainers.image.licenses', which is described as 'License(s) under which contained software is distributed as an SPDX License Expression'.[41] But a typical container image is built from hundreds of components. An SPDX licence expression is generally used to convey licensing information for a single source file, or perhaps a simplified view of the licensing of one package. Such expressions are designed to allow composite expressions, such as 'GPL-2.0-or-later OR BSD-3-Clause'.[42] The SPDX licence's (see Chapter 7) expression format is not designed to represent the complex licensing information details that would

[38] For further information on the registry-native approach, including current and anticipated future implementation details, see Peterson, 'Making compliance scalable in a container world, see note 35.

[39] See, e.g., <https://compliance.linuxfoundation.org/about/> (Linux Foundation Open Compliance Program), accessed 10 August 2020.

[40] <https://github.com/opencontainers/image-spec> accessed 10 August 2020.

[41] <https://github.com/opencontainers/image-spec/blob/master/annotations.md#pre-defined-annotation-keys> accessed 10 August 2020.

[42] See SPDX Specification v2.2.0, Appendix IV: SPDX License Expressions, available at <https://spdx.github.io/spdx-spec/appendix-IV-SPDX-license-expressions/> accessed 10 August 2020.

accurately describe a typical container image, which would require an extremely lengthy conjunctive expression of standardised and ad hoc licence identifiers of little communicative value to those concerned about identifying issues of substantive Open Source licence compliance. There is some evidence that the practice of using Dockerfiles has led to confusion over how to describe the licence of a container image accurately. A Dockerfile is a text document that contains a set of commands for assembling a container image.[43] Although Dockerfiles tend to be fairly trivial and non-expressive, Dockerfiles are sometimes placed under an Open Source licence through application of a licence notice at the top of the file. In those cases, it sometimes happens that the nominal Dockerfile licence is mistakenly assumed to be the licence of the set of container images that can be generated from the Dockerfile. In actuality, a Dockerfile might be licensed under the MIT licence, while the associated container image will generally have some complex composite licence that will include GPLv2 and GPLv3.[44]

Another misconception is that the licence of a container image can be determined through reverse engineering of the image to attempt to extract licence texts. This is not a reliable approach because there is no reason to assume that the image has compliantly included all licence texts and of course reverse engineering is not legally permissible in all jurisdictions, though reverse engineering is implicitly allowed under Open Source licences and therefore is allowed in cases where a binary file is actually governed by an Open Source licence. Licence files have sometimes been deliberately removed from container images in an effort to make them smaller.

The best way to discover accurate information about the licensing terms applicable to a container image is to access the corresponding source code of the components of the image. This is another argument in favor of the registry-native approach discussed in the previous section. With complete source code, scanning tools like ScanCode Toolkit[45] can be used to extract the kinds of legal information that the user considers important.[46]

20.5 Containers and Network Services Copyleft

For the most part, copyleft licence requirements, including source code obligations and restrictions on licensing of derivative works, are triggered by

[43] 'Best practices for writing Dockerfiles', <https://docs.docker.com/develop/develop-images/dockerfile_best-practices/> accessed 10 August 2020.

[44] Hemel, see note 11.

[45] <https://github.com/nexB/scancode-toolkit> accessed 10 August 2020.

[46] Peterson, 'Making compliance scalable in a container world', see note 35; see also Peterson, 'The source code is the license' (29 December 2017), <https://opensource.com/article/17/12/source-code-license> accessed 10 August 2020.

distribution of copies to third parties. This is true of the GPL and GNU LGPL in the GPL family, as well as less widely adopted copyleft licences like the Eclipse Public Licence and Mozilla Public Licence. The growth of web services-based applications in the early 2000s gave rise to a concern that the GPL and other existing copyleft licences had a web services 'loophole'.[47] Then as now, web services were commonly based in part on upstream Open Source software, and improvements to that upstream code were not compelled by traditional copyleft licences to be shared as Open Source-licensed source code because such improvements were generally not distributed to third parties in source or binary form.[48]

This concern, along with an almost entirely distinct interest among some vendors in experimentation with copyleft/proprietary dual-licensing business models, led to development of a class of copyleft licences that attempted to extend the trigger for copyleft licence compliance to non-distribution scenarios involving provision of network services to third parties. Most of these licences effectively attempted to redefine 'distribution' to capture the network services case;[49] these licences are little used today, if not altogether obsolete. The one network services copyleft licence that has survived this earlier period of experimentation is the GNU Affero General Public Licence version 3 (AGPLv3), an FSF-maintained variant of GPLv3 first released in 2008.[50]

Instead of defining distribution or 'conveying', AGPLv3 implements the extension of copyleft to the deployment of network services using the defined term 'modified version' (which in GPLv3 is a synonym for 'work based on the Program'):[51]

> Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software.[52]

[47] See Tim O'Reilly, 'The GPL and Software as a Service' (12 July 2007), <http://radar.oreilly.com/2007/07/the-gpl-and-software-as-a-serv.html> accessed 10 August 2020.

[48] See Richard Fontana, 'The AGPL, Network Copyleft and the Cloud', <https://www.youtube.com/watch?v=ncXa0LFeQY0> accessed 21 August 2022 (presentation given for OpenUK Future Leaders session).

[49] See, e.g., Open Software Licence 3.0, available at <https://opensource.org/licenses/OSL-3.0> accessed 10 August 2020 (defining 'External Deployment' as 'the use, distribution, or communication of the Original Work or Derivative Works in any way such that the Original Work or Derivative Works may be used by anyone other than You, whether those works are distributed or communicated to those persons or made available as an application intended for use over a network" and requiring that "You must treat any External Deployment by You of the Original Work or a Derivative Work as a distribution" ').

[50] <https://www.gnu.org/licenses/agpl-3.0.en.html> accessed 10 August 2020.

[51] See note 18.

[52] AGPLv3 § 13.

AGPLv3 is sometimes described as a licence designed for 'cloud', but this illustrates the ambiguity surrounding that technology industry term. Particularly in popular discussion of consumer IT issues, 'cloud' is often used as a synonym for software-as-a-service (SaaS), which itself can be seen as equivalent to providing software functionality remotely through network services. AGPLv3 is indeed designed to capture the SaaS case. In enterprise computing, however, 'cloud computing' tends to refer to running workloads within 'clouds', which are understood to mean public, private, and hybrid IT environments that abstract, pool, and share scalable resources across a network.[53] The two notions of 'cloud' overlap, because SaaS, including familiar large-scale consumer web services, is often implemented today using cloud computing techniques, increasingly through the use of containers, container orchestration, and other methods of cloud native development. It is however entirely possible to implement SaaS without the use of cloud computing in this specialised sense.

While AGPLv3 is the one notable network services licence to have survived the earlier period of copyleft licence experimentation in the 2000s, it has never been widely adopted for Open Source projects. Nevertheless, it is conceivable that AGPLv3-licensed packages will be included in a container. This may give rise to a concern that running a containerised application that includes an AGPLv3 package in production to provide a public-facing web service—a common use case for containers—will give rise to copyleft obligations, including a requirement to make source code available to service users.

If the AGPLv3 software is used without modification, which will be typical for containers since containers largely reuse pre-built upstream packages without change, then the network services copyleft requirements of AGPLv3 section 13 are not triggered.[54] In the rarer case in which the container image creator has made modifications to the AGPLv3-licensed software, there may be a requirement to provide the 'Corresponding Source' of that software to users, if indeed users are 'interacting with it remotely', which will not inherently be the case merely because a container including such AGPLv3-licensed software is used in providing the service. However, the scope of copyleft in such cases will not extend to 'mere aggregation' within a container and across multiple containers, for all the same reasons that were given in the discussion of the container image distribution case earlier.

20.6 The Rise of 'Source-Available' Licences Targeting Cloud Service Providers

Modern cloud computing, including its current container-driven cloud native phase, has been enabled by a rich ecosystem of Open Source projects and thus

[53] 'What is cloud computing?', <https://www.redhat.com/en/topics/cloud> accessed 10 August 2020.

[54] See Jeffrey Robert Kaufman, 'Do I need to provide access to source code under the AGPLv3 license?' (18 January 2017), <https://opensource.com/article/17/1/providing-corresponding-source-agplv3-license> accessed 10 August 2020.

by Open Source licensing. In the context of projects providing libraries and tools around web and cloud technology, it has long been observed that Open Source is increasingly dominated by non-copyleft licensing.[55] The evolution of Open Source licensing itself does not seem to have been significantly influenced by the industry shift to cloud. AGPLv3, a product of the pre-cloud era targeting the 'web services loophole', has not been widely adopted. In the years since the introduction of AGPLv3, only one Open Source network services copyleft licence of note has appeared: the Cryptographic Autonomy Licence 1.0 (CAL), which was approved by the Open Source Initiative (OSI) in 2020.[56] The novel features of CAL are its 'User Autonomy' provisions, rather than its mechanism for extending copyleft to network services; in any case, CAL has not yet seen notable adoption beyond its initial use case in the Holochain project.[57]

In recent years, much attention has been given to the phenomenon of large third-party cloud service providers monetising popular Open Source projects through offering proprietary 'wrapper' services that benefit from operational advantages.[58] Companies maintaining such projects (which in many cases involve database technology),[59] generally using some variety of 'Open Core' business model, have decried what they regard as 'strip mining' of their projects by cloud providers. Several of these companies have reacted to this development by migrating their projects to 'source-available' licences, so called because they are applied to published source code and may resemble Open Source licences in certain respects, but they do not conform to the software freedom norms underlying the OSI's Open Source Definition (OSD).[60] These source-available licences are designed more or less explicitly to limit or prevent cloud providers from offering competing services based on the same projects, something which is definitionally allowed under any Open Source licence. Some commentators have described these licences as 'hybrid' or 'non-compete' licences.[61]

The source-available non-compete licences deviate from Open Source licensing norms in various ways. The Commons Clause,[62] which Redis Labs used for

[55] See Christine Lemmer-Webber, 'A Field Guide to Copyleft Perspectives' <http://dustycloud.org/blog/field-guide-to-copyleft/> accessed 18 April 2022.

[56] <https://opensource.org/licenses/CAL-1.0> accessed 18 April 2022.

[57] <https://github.com/holochain/holochain> accessed 18 April 2022.

[58] See Stephen O'Grady, 'Tragedy of the Commons Clause' <https://redmonk.com/sogrady/2018/09/10/tragedy-of-the-commons-clause/> accessed 18 April 2022.

[59] See Josh Berkus, 'Why database projects can't leave licenses alone', <https://www.youtube.com/watch?v=uoKcueUkCX4&feature=youtu.be> accessed 18 April 2022 (presentation given at State of the Source Summit 2020).

[60] <https://opensource.org/osd> accessed 18 April 2022 (including requirements that licences not discriminate against persons or groups and fields of endeavour); see also 'What is free software?', <https://www.gnu.org/philosophy/free-sw.en.html> accessed 18 April 2022 (FSF's Free Software Definition).

[61] Stephen O'Grady, 'What does Open Source mean in the era of cloud APIs?', <https://redmonk.com/sogrady/2019/01/25/Open Source-cloud-apis/> accessed 18 April 2022.

[62] <https://commonsclause.com/> accessed 18 April 2022.

previously AGPLv3-licensed Redis modules before more recently switching them to the Redis Source Available Licence,[63] supplements a standard Open Source licence with a refusal to grant 'the right to Sell the Software'. The Redis Source Available Licence grants permission to 'use' the software 'only as part of Your Application, but not in connection with any Database Product that is distributed or otherwise made available by any third party'. CockroachDB uses a variant of the Business Source Licence that prohibits the licensee from providing CockroachDB as a service.[64] The Timescale Licence, which covers certain parts of Timescale Community Edition, prohibits using the software to provide a database as a service.[65] The Fair Source Licence, created by Sourcegraph, requires organisational licensees to pay for a licence once they exceed a specified threshold of users.[66] The Confluent Community Licence prohibits 'making available any software-as-a-service, platform-as-a-service, infrastructure-as-a-service or other similar online service that competes with Confluent products or services that provide the Software'.[67]

The Server Side Public Licence version 1[68] (SSPL) deserves closer attention. SSPL differs from other source-available non-compete licences in two respects: it is based on an Open Source licence text and its licence steward sought to have the licence legitimised as an Open Source licence. SSPL was created in 2018 by MongoDB, Inc. for its MongoDB Community Server, which was previously licensed under AGPLv3. In 2021 it was adopted by Elastic for its Elasticsearch and Kibana projects as a replacement for the Apache Licence 2.0.[69]

Like AGPLv3, SSPL largely reuses the GPLv3 text with few changes. However, in place of AGPLv3 section 13, SSPL has the following provision:

> 13. Offering the Program as a Service.
>
> If you make the functionality of the Program or a modified version available to third parties as a service, you must make the Service Source Code available via network download to everyone at no charge, under the terms of this License. Making the functionality of the Program or modified version available to third parties as a service includes, without limitation, enabling third parties to interact

[63] <https://redislabs.com/wp-content/uploads/2019/09/redis-source-available-license.pdf > accessed 18 April 2022.

[64] Peter Mattis, Ben Darnell, and Spencer Kimball, 'Why we're relicensing CockroachDB', <https://www.cockroachlabs.com/blog/oss-relicensing-cockroachdb/> accessed 18 April 2022.

[65] <https://www.timescale.com/legal/licenses> accessed 18 April 2022.

[66] <https://fair.io/?a> accessed 18 April 2022.

[67] <https://www.confluent.io/confluent-community-license> accessed 18 April 2022.

[68] <https://www.mongodb.com/licensing/server-side-public-license> accessed 18 April 2022.

[69] Shay Banon, 'Doubling down on open, Part II', <https://elastic.co/blog/licensing-change> accessed 18 April 2022.

> with the functionality of the Program or modified version remotely through a computer network, offering a service the value of which entirely or primarily derives from the value of the Program or modified version, or offering a service that accomplishes for users the primary purpose of the Program or modified version.
>
> 'Service Source Code' means the Corresponding Source for the Program or the modified version, *and the Corresponding Source for all programs that you use to make the Program or modified version available as a service, including, without limitation, management software, user interfaces, application program interfaces, automation software, monitoring software, backup software, storage software and hosting software, all such that a user could run an instance of the service using the Service Source Code you make available.*[70]

The key difference between SSPL section 13 and AGPLv3 section 13 is that the AGPLv3 Corresponding Source requirement covers only the 'modified version' as that is defined in GPLv3 and AGPLv3. The source code that must be offered to interacting users is equivalent to what would be required under GPLv3 if an object code version were being distributed. In SSPL, the required source code extends to the entire stack of programs used to implement the service, including programs that would be considered 'mere aggregation' of separate and independent programs under the GPL. Moreover, the entire set of Corresponding Source, including the source code of all those independent programs, must be provided under SSPL. In practice this requirement is not possible to comply with (without avoiding providing a service entirely), because it is not practically possible to form an entire services stack out of SSPL-licensed components without implementing much of that stack from scratch. For this reason, the SSPL section 13 requirement can be regarded as a prohibition on providing the software as a service, disguised as a modification of the GPLv3/AGPLv3 Corresponding Source requirement.

Unlike the licence stewards of other source-available non-compete licences, MongoDB submitted SSPL for approval by the OSI in 2018. The licence was discussed over the course of several months on the OSI's license-review mailing list, during which MongoDB submitted a revised draft; reception was largely hostile. MongoDB withdrew the licence from consideration, likely to avoid formal rejection by the OSI.[71] Objections to the licence from an OSD conformance perspective generally focused on the anti-discrimination provisions of the OSD as well as the spirit, if not the letter, of OSD 9 ('The license must not place restrictions on other software that is distributed along with the licensed software. For example, the licence must not insist that all other programs distributed on the same medium must be Open Source software.').

[70] Server Side Public Licence § 13 (emphasis added).

[71] OSI Board of Directors, 'The SSPL is not an Open Source license' (19 January 2021) <https://opensource.org/node/1099> accessed 18 April 2022.

The use of source-available licences has been the subject of much recent controversy in the Open Source technical community, not just because these licences violate Open Source definitional norms and have been applied to widely used projects, but also because they are seen as creating confusion around the meaning of Open Source as an identifying label. Some have complained that the 'Open Source companies' at issue have used Open Source licensing as a means of gaining adoption and thus causing widespread dependency on the projects, raising the costs to users of switching to alternative Open Source technologies. At the same time, the effectiveness of the source-available licence strategy as a protection against competition from cloud providers is doubtful, given the ability of well-resourced cloud providers to reimplement, fork, or create application programing interface- (API) compatible alternatives to such projects.[72]

[72] Stephen O'Grady, 'Cockroach and the source available future', <https://redmonk.com/sogrady/2019/06/21/cockroach-source-available> accessed 18 April 2022.

21
Public Sector and Open Source

Iain G Mitchell KC

21.1 Introduction

At publication of the first edition, although Open Source software was gaining in importance, the Microsoft/Apple duopoly in operating systems appeared to have acquired a very substantial stifling effect on the operating system market as discussed in Chapter 17. It took a lot of nerve for users to swim against the tide of Microsoft Office, and anyone who has tried to create a document either in Open Source or a proprietary program other than MS Word, save it as a .doc or .docx file, send it to a recipient, who then tried to open it in MS Office will know that this was at best a hit and miss operation. A plea to the recipient to install Open Office seemed to fall on deaf ears and, on balance, it all seemed so much easier for the author of the document just to go out and purchase a licence for MS Office.

Since then, a number of factors has created space for Open Source software to flourish.

First, there has been some accommodation of Open Source by the dominant players. Microsoft Word 2010 introduced native support for the .odt format, though both user unfamiliarity and continued use by some users of outdated MS office software means that interoperability problems continue to occur.

Iain G Mitchell KC, *Public Sector and Open Source* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Iain G Mitchell KC 2022. DOI: 10.1093/oso/9780198862345.003.0021

Also, both Microsoft and Apple have sought to some degree to engage with Open Source,[1] as discussed in Chapter 10, but, undoubtedly, the major driver has been a conscious shift (at least in the case of Microsoft) in its business model, both in the move to seeing itself as a supplier of software as a service and to participating consciously not just as a user but as a contributor to Open Source, indeed one of the biggest in the world. These shifts reflect a major shift in the market towards cloud computing: what matters is the service sold, more than the software used to provide the cloud service, although it is generally recognised that the cloud is built on Open Source software.

Companies with a dominant position can be tempted to abuse it, as discussed in Chapter 17, and whilst enforcement action can rein in the worst excesses, it is business self-interest which drives the shift from being a minority enthusiasm for geeks to achieving a significant market share. But the forces which maintain the main player's market dominance can often be those same forces which stop a breakthrough for competitors.

Historically, Open Source's clear economic advantages over proprietary software caused a gain in market share, despite previous bias towards proprietary software. To achieve its full potential both requires a level playing field and a sufficiently large user base. Governments have a clear role to play in the creation of a level playing field, but also, and less obviously, through the medium of public procurement, have the opportunity to achieve it. Public procurement accounts for, on average, about 12 per cent of the gross domestic product of the Organisation for Economic Co-operation and Development (OECD) countries.[2] It is clear that, on any view, the public sector is a major player in IT procurement—large enough to have a significant influence on the respective market shares of Open Source and proprietary software. The creation of a 'critical mass' of public sector customers serves to encourage private sector buyers.

The public sector can choose to be proactive in its support for Open Source, or merely to be neutral so that there are, at least, no barriers to the procurement of Open Source, or, to demonstrate a bias towards particular proprietary solutions. Whatever the policy, there are both obligations and limits on what a public authority can do as part of the procurement process because public procurement sits within a framework of international and local laws.

21.2 The International Context—The WTO

At the international level, the World Trade Organization (WTO) recognises the key role which public procurement has to play in stimulating world trade. It

[1] See, e.g., Matt Asay, 'Apple is doubling down on Open Source' Tech Republic, 9 November 2016, <https://www.techrepublic.com/article/apple-is-doubling-down-on-open-source/> accessed 14 January 2020.

[2] OECD website, <https://www.oecd.org/gov/public-procurement/> accessed 14 January 2020.

states on its government procurement gateway that open, transparent, and non-discriminatory procurement is generally considered to be the best tool to achieve value for money but refers also to the use by many governments of procurement to achieve domestic policy goals. It warns against preferential treatment for domestic goods, services, and suppliers which discriminates against foreign suppliers and therefore acts as a trade barrier in this sector.[3]

Notwithstanding the undoubted importance of government procurement, it was excluded from the General Agreement on Tariffs and Trade (GATT), and from the main market access commitments of the General Agreement on Trade in Services (GATS'. The WTO has sought to plug that gap, with the 1979 first Agreement on Government Procurement (GPA), which has since been twice renegotiated, the most recent, 2014 version, currently having twenty signatories, of which one is the European Union (EU) on behalf of its twenty-seven member states.[4]

The Agreement is plurilateral, not binding all members of the WTO but only those which have chosen to accede to the Agreement, and it does not cover all forms of government procurement in all signatory states.[5]

21.3 The European Procurement Law Context

Although a full exposition of public procurement law does not fall within the scope of this book, it is useful to have a high level understanding of the underlying principles in order to understand some of the issues surrounding software procurement.[6]

The EU acts within the terms of its Treaty competences which include Article 3(3) of the Treaty on European Union (TEU) which deals with the Single Market. The Union is given power 'to adopt measures with the aim of establishing or ensuring the functioning of the internal market, in accordance with the relevant provisions of the Treaties'.[7] Title II makes provision in respect of the Free Movement of Goods and Title IV for the Free Movement of Persons, Services, and Capital.

[3] WTO, 'Government procurement' <http://www.wto.org/english/tratop_e/gproc_e/gproc_e.htm> accessed 14 January 2014.

[4] The other signatories are Armenia, Australia, Canada, Hong Kong, Iceland, Israel, Japan, Korea, Liechtenstein, Moldova, Montenegro, the Netherlands with respect to Aruba, Norway, Singapore, Switzerland, Chinese Taipei, Ukraine, and the US. A further ten members are in the process of acceding.

[5] WTO <https://www.wto.org/english/tratop_e/gproc_e/gp_gpa_e.htm> accessed on 14 January 2020

[6] See, generally, Christopher Bovis, *EU Public Procurement Law*, 2nd edn (Oxford: Oxford University Press, 2015); Albert Sanchez-Graells, *Public Procurement and the EU Competition Rules*, 2nd edn (Oxford: Hart Publishing, 2015); and Peter Trepte, *Public Procurement in the EU: A Practitioner's Guide*, 2nd edn (Oxford: Oxford University Press, 2007).

[7] Part 3, Title I (arts 26 and 27) of the Treaty on the Functioning of the European Union (TFEU).

The ways in which the Union exercises its competences include Regulations, which are immediately effective throughout the EU, and Directives, which require member states to transpose their provisions into their respective national laws.

In the area of public procurement, the first consolidation of procurement rules came with Directive 71/305 co-ordinating procedures for the award of public works contracts and Directive 77/62 in relation to public supply contracts, subsequently supplemented, amended, and replaced by Directive 2004/18/EC ('The Public Sector Directive');[8] Directive 2004/17/EC ('The Utilities Directive');[9] and Directive 2007/66/EC ('The Remedies Directive'),[10] collectively referred to as the Procurement Directives. These were replaced in 2014 by a new set of Directives: Directive 2014/24/EU (the 'Public Procurement Directive');[11] Directive 2014/25/EU ('the Utilities (Sectors) Directive');[12] and Directive 2014/23/EU ('the Concessions Directive').[13]

Article 4 of the Public Procurement Directive, Article 15 of the Utilities (Sectors) Directive, and Article 8 of the Concessions Directive set minimum thresholds below which the Directives do not apply and which, at the time of writing, are €5,350,000.[14] However, that does not mean that contracts of a lower value fall outside the European public procurement regime.

At the heart of the Single Market is the abolition of both customs barriers and non-tariff barriers, such as differing national standards, and the creation of a level playing field on which businesses from all of the member states can compete fairly. At the time of the introduction of the original Procurement Directives[15] there was a concern that public authorities could make direct awards of contracts, cutting would-be contractors out of the opportunity to bid. The Procurement Directives made clear that they aimed to improve the access of service providers to procedures for the award of contracts. Public sector contracts are also governed by both

[8] Directive 2004/18/EC of the European Parliament and of the Council of 31 March 2004 on the coordination of procedures for the award of public works contracts, public supply contracts and public service contracts [20 April 2004] OJ L134/114.

[9] Directive 2004/17/EC of the European Parliament and of the Council of 31 March 2004 coordinating the procurement procedures of entities operating in the water, energy, transport and postal services sectors [30 April 2004] OJ L134/1.

[10] Directive 2007/66/EC of the European Parliament and of the Council of 11 December 2007 amending Council Directives 89/665/EEC and 92/13/EEC with regard to improving the effectiveness of review procedures concerning the award of public contracts [20 December 2007] OJ L335/31.

[11] Directive 2014/24/EU of the European Parliament and of the Council of 26th February 2014 and repealing Directive 2004/18/EC on Public Procurement [28 March 2014] OJ L94/65

[12] Directive 2014/25/EU of the European Parliament and of the Council of 26 February 2014 on procurement by entities operating in the water, energy, transport and postal services sectors and repealing Directive 2004/17/EC [28 March 2014] OJ L94/243

[13] Directive 2014/23/EU of the European Parliament and of the Council of 26 February 2014 on the award of concession contracts [28th March 2014] OJ L94/1

[14] This is the normal threshold, though there are lower thresholds stipulated for certain specified types of contract.

[15] Directives 92/50/EEC (Public Services Contracts), 93/36/EEC (Public Supply Contracts), and 93/37/EEC (Public Works Contracts).

the principles laid down in the Treaties, and the principles derived from the Treaty principles, namely:

- equal treatment;
- non-discrimination;
- mutual recognition;
- proportionality; and
- transparency.

The Public Procurement Directive requires for contracts above threshold that contracting authorities should treat economic operators equally and without discrimination and should act in a transparent and proportionate manner.

The European Court of Justice (ECJ), in *Telaustria Verlags GmbH and Telefonadress GmbH v Telekom Austria AG*,[16] applied these principles not only to above-threshold contracts but also to most below-threshold contracts.

The Court expanded upon this in *Bent Mousten Vestergaard v Spøttrup Boligselskab*,[17] stating that 'the mere fact that the Community legislature considered that the strict special procedures laid down in those directives are not appropriate in the case of public contracts of small value does not mean that those contracts are excluded from the scope of Community law'. Accordingly, below-threshold contracts are subject to the Treaty principles, even although not falling within the scope of the Procurement Directives. Later cases involving below-threshold contracts followed, including *Parking Brixen GmbH v Gemeinde Brixen & Stadtwerke Brixen AG*[18] and *Medipac-Kazantzidis AE v Venizeleio-Pananeio*.[19]

However, since the Treaties aim to create a Single Market by ensuring competition across borders, for the Treaty principles to govern a procurement exercise, there has to be a cross-border interest:[20]

> Setting a financial threshold above which contracts are subject to public procurement directives is based on a single premise, that contracts of small value do not attract operators established outside national borders; such contracts are thus devoid of Community implications. However, that rebuttable presumption is open to evidence to the contrary. . . .[21]

[16] Case C-234/98 [2000] ECR 1-10770.

[17] Case C-59/00 [2001] ECR 1-09505.

[18] Case C-458/03 [2005] ECR 1-08585.

[19] Case C-6/05 [2007] ECR 1-04557.

[20] *Consorzio Aziende Metano (Coname) v Comune di Cingia de'Botti* at (Case C-231/03 [2005] ECR 1-07287)§ 20.

[21] See the Advocate General in the conjoined cases of *SECAP SpA* and *Santorso Soc. Cooparl v Comune di Torino* (Cases C-147/06 and 148/06 [2008] ECR 1-03565) at § 23.

In its judgment, the court affirmed that, although the strict procedures laid down in the Directives did not apply to below-threshold contracts, contracting authorities were held still to be bound to comply with the fundamental rules of the Treaty, including the principle of non-discrimination on the ground of nationality where the contracts concerned are of cross-border interest. It is for the contracting authority to make the initial assessment, though this is open to judicial review.[22]

> The approach of the court is to regard public sector contracts as falling into two classes: those where there is and those where there is not a cross-border interest. It may however be helpful to treat public sector contracts as, in effect, falling into three classes: those which are of such a negligible value that no cross-border interest is likely to arise; those (not falling in the first category) where the contract value falls below threshold, and in which there is a rebuttable presumption that no cross-border interest arises, but if that presumption is rebutted, then the Treaty principles and derived principles are applicable; and those in which the contract value is above threshold, with a[n] irrebuttable presumption that there is a cross-border interest, which are subjected to the strict and detailed rules set forth in the Procurement Directives.[23]

Those which are above threshold clearly fall into the first category (cases where there is a cross-border interest); those which are below threshold might fit into either category depending upon the view taken as to whether the presumption that there is not a cross-border interest is overcome by the facts, and the responsibility for making the judgment of whether a cross-border interest arises is for the public body concerned, though any such decision is always open to judicial review.[24]

Assuming that there is a cross-border interest and whether or not the contract value is above threshold, the public authority will be required to conduct the procurement exercise in accordance with the Treaty and derived principles, including the principles of equal treatment, non-discrimination, mutual recognition, proportionality, and transparency.

The 2014 Public Sector Directive replicated the procurement procedures set out under the 2004 Directive, and this is likely to be the normal mode of procurement in the case of an above-threshold contract. However, the 2014 Public Sector

[22] *Coname* at paragraph 30.

[23] For a more detailed analysis of the jurisprudence of the ECJ in this matter, see the submissions by the present author as Counsel in *Sidey Ltd v Clackmannanshire Council and Pyramid Joinery and Construction Ltd* 2010 SLT 607, at §§ 22 *et seq*.

[24] *SECAP*, see note 21; *R (on the application of Chandler) v Secretary of State for Children, Schools and Families* (2010) CMLR 19; *Sidey Ltd v Clackmannanshire Council* 2012 SLT 334. In the Petition for Judicial Review (reported 2012 SLT 334) which followed the case just cited, the court determined on the facts that, in a below-threshold contract, no cross-border interest had been engaged, even although the Council had voluntarily conducted the tendering exercise in question according to the rules for above-threshold contracts.

Directive also introduced several innovative new modes of procurement, including Design Contests,[25] and Innovation Partnerships.[26]

The Innovation Partnership regime enables contracting authorities, where they 'identify the need for an innovative product, service or works that cannot be met by purchasing products, services or works already available on the market'[27] to set up an innovation partnership with one or more partners. The Partnership may be structured in successive phases in order to develop an innovative product or service, and the contracting authority may purchase of the resulting output provided that it corresponds to the performance levels and maximum costs agreed between the contracting authorities and the participants.[28] This enables greater innovation through increased flexibility.

Under the 2004 Public Sector Directive, the criterion for selection was either lowest price or most economically advantageous tender. The 2014 Directive innovated on this by making the most economically advantageous tender the sole basis, and expanded the criteria for tender assessment to include qualitative, environmental, or social aspects, so long as they are linked to the subject matter of the contract.

How, then, does this legal structure impact upon questions of software procurement?

21.4 Issues in Software Procurement

21.4.1 The policy setting of the directives

The 2014 Public Sector Directive provides a hopeful setting in which those who have innovative solutions can compete on a level playing field. Though not restricted to Open Source, plainly Open Source developers are well placed to benefit from the more benign ecosystem.

The Directive puts explicit emphasis on innovation, stating[29] that research and innovation, including eco-innovation and social innovation, are among the main drivers of future growth, and it urges public authorities to make strategic use of public procurement to spur innovation.

There are two important qualifications.

First, innovation is encouraged, but, of course, is not obligatory.

Second, the principles, mechanisms, and rules in the Directive apply only where there is a procurement exercise which falls within its ambit.

[25] Arts 78–82.

[26] Art 31.

[27] Art 31(1).

[28] Art 31(2).

[29] Recital 47.

What then, are the sorts of issues which are likely to arise when considering the interface between public procurement and Open Source?

21.4.2 Is there procurement at all?

It is important to stress that the EU procurement regime applies to procurement by the public sector, which is defined in Article 1(9) of the Public Sector Directive as including 'the State, regional or local authorities, and bodies governed by public law'.

The definition of a body governed by public law is of considerable importance, encompassing, as it does, bodies which do not have an industrial or commercial *character*, placing such bodies within the ambit of the EU procurement regime, whereas a body not having a commercial or industrial *purpose* would not be regarded as an 'enterprise' and so not fall within the ambit of competition law.[30]

The procurement regime does not apply to the private sector. If a private enterprise in its procurement activities, whether in connection with IT purchases or anything else, is opaque and treats potential suppliers unequally, then whatever other remedies there may be for an aggrieved potential tenderer, there are none under the European procurement regime.

It also does not apply if there is no procurement. For there to be a procurement exercise at least two separate parties dealing with each other are required. If the public authority is developing its own software then securing that software for its use is not a procurement exercise at all. This applies even if the departments involved are organisationally separated.

This is extended under Article 12 of the Public Procurement Directive which specifically exempts contracts between certain entities in the public sector and lays down a number of conditions all of which must be met before the exemption applies, including that the contracting authority should exercise control (as defined) over the other party. Specific contracts are also excluded from the Public Procurement Directive, at Part 3 of the Directive. Some are excluded simply because they are included in the other Directives.[31] Also excluded are 'public contracts and design contests for the principal purpose of permitting the contracting authorities to provide or exploit public communications networks or to provide to the public one or more electronic communications services',[32] and certain

[30] See the conjoined cases of Cases C-159/91 *Poucet* and C-160/91 *Pistre* [1993] ECR I-637; Case C-67/96 *Albany* [1999] ECRI-5751; and Case T-319/9 *Nacional de Empresas de Instrumentación Científica, Médica, Técnica y Dental (FENIN) v Commission of the European Communities* [2003] 5 CMLR 1 and, on appeal, Case C-205/03 [2006] 5 CMLR 7.

[31] For example, art 7 excludes from the ambit of the Public Procurement Directive the contracts which fall within the scope of the Utilities (Sector) Directive.

[32] Art 8. These are subject to separate regulation under Directive 2002/21/EC of the European Parliament and the Council of 7 March 2002 on a common regulatory framework for electronic

contracts in the defence and security sector where 'the essential security interests of a Member State cannot be guaranteed by less intrusive measures'[33] or which are declared to be secret or requiring special security measures.[34]

Article 15 presumes that Defence and security sector contracts normally fall within the ambit of the Directive, so the exception for essential security interests might shut certain potential contractors out of the market. An example, albeit from the telecommunications sector, of the sort of issues which may be at play can be found in the controversy during 2019 on whether any economic operator who works with or uses the technology of the Chinese company, Huawei, should be permitted to participate in the development of the UK 5G network.[35] If a contracting authority wishes to acquire software, it might do so as part of a larger package, for example involving services provided by a supplier who uses Open Source software or implements an Open Source solution which would fall within the scope of the Directives. There may also be cases where the contracting authority wishes simply to acquire the software either with a view to developing the software itself or simply using it 'as is' and might choose to buy the software, or simply download it for free. If it chooses to download it, then, since it is not buying anything, there is no 'public contract' within the meaning of Article 2(1)(5) of the Public Procurement Directive[36] and no procurement exercise falling within the ambit of EU procurement law.

Paragraph 2.2.4 of the Interoperable Delivery of European eGovernment Services to Public Administrations, Businesses and Citizens (IDABC) *Guideline on Public Procurement of Open Source Software*[37] gives guidance on best practice for downloading, recommending that the public authority should first determine its acquisition requirements and then should download the relevant software itself without fee as part of that already-determined acquisition process, whilst simultaneously issuing an invitation to tender for commercially provided services and support, where required. The *Guideline* stresses that the downloading of the software should be seen as, in effect, an alternative to the step of publishing a tender

communications networks and services 2002 OJ L108/33 (the Electronic communications Framework Directive).

[33] Art 15 (2).

[34] Art 15 (3).

[35] See, e.g., *Mail Online* 'Huawei fires back at Malcolm Turnbull after he warned Boris Johnson not to use the Chinese tech giant for its 5G rollout over security fears' <https://www.dailymail.co.uk/news/article-7889039/Chinese-firm-Huawei-slams-Malcolm-Turnbulls-5G-security-warning.html> accessed 15 January 2020

[36] Art 2(1)(5): ' " 'public contracts" means contracts for pecuniary interest concluded in writing between one or more economic operators and one or more contracting authorities and having as their object the execution of works, the supply of products or the provision of services'.

[37] European eGovernment Services, '*Guideline on Public Procurement of Open Source Software*' (March 2010) <https://joinup.ec.europa.eu/sites/default/files/document/2011-12/OSS-procurement-guideline%20-final.pdf> accessed 15 January 2020.

for the supply of software, and counsels against the unsupervised downloading of software by individuals within the organisation.

For those interested in the future of Open Source, this is something of a double-edged sword as an informed public authority would be able to acquire the Open Source that it wants, without having to be concerned about possible challenges from disappointed would-be tenderers, but if a public authority embarks on a public procurement exercise, it may end up with a proprietary solution, simply because there is no Open Source supplier in the race; no-one would go to the cost and trouble of submitting a tender with a nil value. Of course there may be nothing to stop a free software developer tendering at an economic price by bundling in software support services, even though the public authority might have downloaded the same software for free and it may be that most software acquisitions by public authorities involve some element of added value, even if it is only installing, testing, and maintaining the solution.

21.4.3 Cross-border interest

Neither the TEU nor TFEU principles apply to public procurement where there is no cross-border interest. It is for the public authority concerned to make that judgment in the first instance, subject to the possibility of judicial review. The public authority may decide that there is a cross-border interest, or fail to apply its mind to the question, and then proceed to run the procurement exercise according to either the Procurement Directives or the Treaties and derived principles, but risk finding itself the respondent in a judicial review. This can happen when the public authority, in conducting a procurement exercise under the EU regime, infringes any of the principles contained in the Treaties (known as 'Treaty principles'), or principles derived therefrom (known as 'derived principles') and then seeks to defend itself by asserting that there was no cross-border interest in the first place.[38]

Of more concern are cases where the public authority decides that there is no cross-border interest and does not put the contract out to tender. How is the would-be tenderer to know that he missed the opportunity in the first place? Although there is a strict requirement on a public authority, to publish a Contract Notice in the Official Journal in the case of an above-threshold contract, in the case of a below-threshold contract, publication of such a notice is permitted, but not mandated.[39] If the would-be tenderer subsequently comes to find out about a failure to advertise a contract which does engage cross-border interest (whatever the public

[38] This in fact happened in *Sidey Ltd v Clackmannanshire Council* (2012) SLT 334 though the defence succeeded only up to a point.

[39] Art 51 (6).

authority may think), then the courts will not be slow to invoke the principle of transparency to set aside any contract made without sufficient advertising.[40]

21.4.4 An own goal?

Although a detailed survey of the rules governing the assessment of the economic operator's suitability and competence lies outside the scope of this chapter, it is worth mentioning the situation where a would-be tenderer or his employees have been involved in the preparation of the contracting authority's invitation to tender, contract notice, or specification, as in the case of *Fabricom*.[41] Fabricom was prevented from tendering by a Belgian Royal Decree which imposed an absolute ban on persons who had been instructed to carry out research or studies or development works in connection with public works or services in participating in a tender.

The ECJ overturned that decree because it was expressed in absolute terms but recognised the underlying principle of the prohibition of unequal treatment. It stated, at sections 29–31 that a person who had participated in preparatory works may be put at an advantage compared to other tenderers, and there might be a conflict of interest created by his having the opportunity to influence (even unconsciously) the tender conditions so as to favour him. As a result of this, there would be a breach of the principle of equal treatment.

However, the Court also decided that the Royal Decree should be amended to allow for the possibility of a person with such involvement to prove that the experience which he had acquired was not capable of distorting competition. A would-be tenderer should be vigilant for possible involvement by a competitor at an earlier stage, and a contracting authority should be careful not to run the risk accidentally of disqualifying a person whom it might have wished had been free to tender.

This is not to say that a close ongoing cooperation between a public authority and a contractor is impossible. After all, part of the rationale behind the introduction of Innovation Partnerships in the 2014 Directive was to overcome the problem which arose under the previous Directives where, if a public authority partnered with a private company to develop a new and innovative solution, when that solution had been developed, it was necessary to launch a new procurement exercise before awarding a contract to implement that solution. The Innovation Partnership procedure:

[40] *Telaustria*, see note 16; *Parking Brixen*, see note 18.

[41] Cases C-21/03 and C-34/03 *Fabricom SA v Belgium* Cases C-21/03 and C-34/03 [2005] ECR I- 1577.

> should allow contracting authorities to establish a long-term innovation partnership for the development and subsequent purchase of a new, innovative product, service or works provided that such innovative product or service or innovative works can be delivered to agreed performance levels and costs, without the need for a separate procurement procedure for the purchase.[42]

21.4.5 Issues with the specification

21.4.5.1 Introduction

The problems which are most likely to be met with in practice are not so much to do with whether the procurement rules apply at all, or whether a tendering exercise has been sufficiently advertised, but, rather, whether the technical specification has been drawn up in a way which unfairly tilts the playing field, for example, by specifying a particular product (such as 'Microsoft Office') or, indeed, specifying 'Open Source', or producing an apparently open specification which, upon closer examination turns out can be met by only one supplier.

21.4.5.2 Specified products

The law is quite clear that it is forbidden to specify a particular brand or product, without, at least, adding the words 'or equivalent'. The leading case is *European Commission v The Netherlands*,[43] where the tender specification stated that the operating system required was 'UNIX'[44] and the words 'or equivalent' were not added.

The judgment gives a good flavour of the argument advanced by the Netherlands, which attempted to set up UNIX as a kind of industry *de facto* standard so that it was not necessary to add the words 'or equivalent'. The Court, however, noted that parties agreed that UNIX was the name of a specific product and not a standard. Accordingly, the words 'or equivalent' should have been added. Specifying a particular product is clearly impermissible, even if the product in question has achieved sufficient market dominance that the uninformed begin to think of it as some kind of standard.

That said, might the same objection not be taken to specifying 'Open Source'?

This is a more complex issue, but it is best postponed until after a consideration of effectively exclusive technical specifications in general.

[42] Art 49

[43] Case C-359/93 [1995] ECR I-15; see also Case C-59/00 *Bent Mousten Vestergaard v Spøttrup Boligselskab* Case C-59/00 [2001] ECR 1-09505.

[44] A software system developed by Bell Laboratories of ITT (USA) for connecting several computers of different makes.

21.4.5.3 Effectively exclusive technical specifications

Adding the words 'or equivalent' may not suffice if there is no equivalent, and how is equivalence to be judged? What if the technical specification is such that only one potential tenderer is in a position to meet it?

In *Concordia Bus Finland Oy Ab*,[45] which concerned the awarding by the city of Helsinki of a contract for the operating of buses for its public transport system, the contract notice published in the Official Journal stated that the city would select the most economically advantageous tender rather than the lowest price.[46] Where the 'most economically advantageous' criterion is used, the contracting authority selects a number of criteria, allocates a specific number of points to each, and carries out a scoring exercise on the admissible tenders which have been submitted. In the interests of transparency, the authority is required to publish the criteria and points to be allocated to each.

The contract notice specified that additional points would be available for a tenderer who used buses with a specified low level of nitrogen oxide and external noise levels below 77 decibels. The lowest price was offered by Concordia but it was unable to meet those specifications, whereas a rival tenderer, HKL (which belonged to the city), was able to meet these and won the contract. Concordia challenged this award, *inter alia*, on the ground of unequal treatment, before the national courts who, on appeal, referred the matter to the ECJ for a preliminary ruling.

The Court dealt first with the specification, stating that, in determining the most economically advantageous tender, the contracting authority:

> may take criteria relating to the preservation of the environment into consideration, provided that they are linked to the subject-matter of the contract, do not confer an unrestricted freedom of choice on the authority, are expressly mentioned in the contract documents or the tender notice, and comply with all the fundamental principles of Community law, in particular the principle of non-discrimination.

It then applied that principle to the particular facts to find the criteria stipulated in the contract in question to be permissible.

In considering the circumstance that the criteria shut out other tenderers, it came to the conclusion that although the criteria limited the number of possible tenderers, the principle of equal treatment did not prevent the contracting authority from specifying environmental protection as a criterion in a situation where the authority's own transport undertaking was one of the few undertakings capable of meeting the criteria which had been set. This reasoning was developed

[45] Case C-513/99 [2002] ECR I-7251.

[46] As noted earlier, lowest price was an alternative which was permissible under art 53 of the 2004 Public Sector Directive.

in *EVN AG and Weinstrom GmbH v Austria*,[47] which concerned a contract for the supply of electricity which stipulated that a minimum proportion of the electricity should be generated from renewable resources and gave a weighting in the scoring for electricity generated from renewable sources above that minimum.

The Court found this to be permissible, though with the same provisos as had been articulated by the Court in *Concordia Bus*.

Although a contracting authority may set criteria beyond the purely economic, it cannot arbitrarily select those criteria. From the comments of the Court in *EVN*, one might single out the requirements that the criteria should not be such as to confer an unrestricted freedom of choice on the contracting authority and should be linked to the subject matter of the contract and, from the comments in *Concordia Bus*, the requirement that the criteria should be objectively justifiable.

Following *Concordia Bus,* the European Commission issued advice that:

> [under] the EU public procurement rules, contracting authorities may refer to a brand name to describe a product only when there are no other possible descriptions that are both sufficiently precise and intelligible to potential tenderers.[48]

Following this, it would not normally be possible to refer, for example, to 'MS Office or equivalent' or to 'Intel or equivalent' microprocessors in public tenders, and it may be difficult to posit (at least in the IT market) a situation where the qualification would come in to play.

This might all seem relatively hopeful in furnishing arguments to mount a challenge against a procurement exercise where Open Source software providers find themselves shut out by the technical criteria, but there is a danger that the courts will judge reasonableness from the perspective of the contracting authority, rather than on a wider view.

That happened in the Scottish case of *Elekta Ltd v Common Services Agency*,[49] concerning a tendering exercise carried out by the National Health Service (NHS) in Scotland for the replacement of linear accelerators in all of the cancer units in NHS hospitals in Scotland. In order to operate properly, a linear accelerator needs to be controlled by specialised software.

Five out of the six hospitals concerned had systems purchased some years previously from a company, Varian, which kept its code secret and did not publish its communications interface, so other manufacturers' machines could not interoperate with it. The NHS did not wish to replace its software system, even although

[47] Case C-448/01[2003] ECR I-14558.

[48] European Commission release reference IP/06/443 dated 4 April 2006; this is also a reference to Directive 2004/18/EC, Article 23.

[49] 2011 SLT 815.

by far the most expensive part of the entire system was the hardware which the NHS sought to replace.[50]

The technical specification required that the new equipment should be compatible with the existing ARIA system and made certain functional requirements that could not be met without such interoperability. Elekta (and any other potential tenderer), was effectively shut out of the tender process and raised a legal challenge against the tender process on the basis of the Treaties and derived principles, in particular the requirement of equal treatment.

The judge, Lord Glennie, having considered the *Concordia Bus* and *EVN* cases, expressed the view[51] that the contracting authority must be entitled to decide upon the functional requirements which it wishes to satisfy; that the criteria can be satisfied by only one or a limited range of tenderers does not of itself contravene the principle of equality; and the inclusion of these criteria can only be considered discriminatory if they cannot be justified objectively having regard to the characteristics of the contract and the needs of the contracting authority.

The court found, referring to *Concordia Bus*, that the principle of equal treatment was not infringed and that in setting the criteria the contracting authority had not specified a specific product, Varian.

However, the judge commented that the authority operated a rolling replacement program and did not wish to replace its present system, but observed that Elekta's representative had not argued that the criteria were not objectively justifiable, and, on that basis, reached the view that 'no purpose would be served by requiring the defender to invite tenders for something other than what they in fact want'.[52]

This is a classic case of vendor lock-in. The reason why the NHS needed its new and extremely expensive equipment to be interoperable with the Varian system was that because of a decision made in the past, it is locked forever into Varian's system, or at least for so long as it continues to operate a rolling replacement system. Indeed, the present procurement exercise compounded the situation as the hospital which was not locked in was to have its RMS system replaced as well as its linear accelerator, with the inevitable effect that it too would be locked in for the future. Further, it shuts out competition, giving Varian an effective monopoly.

It may be arguable that the case was correctly decided on the basis of the arguments that were presented to the court, but a more hopeful alternative reading is that the case was badly argued.

The key lies in the requirement for objective justifiability of the criteria set against Elekta's incomprehensible decision to decline to argue that the criteria were

[50] The contract value was £21 million, excluding value-added tax.
[51] At para 14.
[52] At para 20.

not objectively justifiable and represents a lost opportunity to advance this area of the law in what, it is suggested, is its natural direction of travel.

There is a similar, though much earlier case in Spain,[53] where the ECJ found that it was permissible to specify that the software should be MS Windows compatible.

One might have some reservations about the reasoning in that case, particularly the view expressed by the Court that one accords greater flexibility to public administrations in software procurement than in other forms of procurement, which scarcely seems justifiable in procurement law but, in view of the requirement for objective verification, it may be that we are left with a decision which is difficult to reconcile with the underlying principles of procurement law.

Much more hopeful, however, is a recent decision of the Tribunal Administratif de Strasbourg[54] in which a company, SAS Anywhere Services, claimed to have been excluded from a procurement process conducted by the Prefect of the Grand Est region. The Region sought to enter into a framework agreement for the development and maintenance of a collaborative platform operated by two regional directorates (the Directorates of Food, Agriculture and Forests, and of the Environment, Planning and Housing). The platform ran on proprietary software sold by Interstis Partenaires and known as 'Interstis', of which neither the source code nor documentation were made available to potential tenderers other than the Interstis Partenaires itself. SAS argued that this made it functionally impossible for them to tender, whereas the Region argued that since actual interaction with the software was not required at the stage of tendering for the framework contract, the absence of such material did not infringe upon the requirement for equal treatment.

The Tribunal decided that the lack of information relating to the functionality of the software led to a lack of equal treatment amongst potential tenderers; as it gave only Interstis Partenaires the information needed to assess the complexity and volume of the source code, notwithstanding that the lodging of a tender did not require any intervention in the software itself before conclusion of the framework agreement.

21.4.5.4 Specifying Open Source

Open Source is not a brand name nor an identifiable product, so including a requirement in a tender specification for the use of Open Source software ought not to cause problems under the *Netherlands* and *Bent Mousten Vestergaard* line of authority. It would not restrict competition to only one supplier or group of suppliers: anyone can seek to tender.

Open Source might be regarded as having a non-technical characteristic, that is the licensing terms under which it is written, or even a technical characteristic; namely the ability of the source code to be used and modified, whereas with

[53] Sentencia del Tribunal Supremo de 6 de Julio de 2004 (Supreme Court decision 6 July 2004).

[54] TA Strasbourg, 16 avr. 2019 n 1901892.

proprietary software the source code may neither be provided nor legally be capable of modification. Viewed in this light, one is firmly in *Concordia Bus* territory, and the issues relate to whether such a requirement can be objectively justified.

Notwithstanding this legal analysis, in practice some doubt appears to have surrounded whether a technical specification may specifically call for 'Open Source'. Under the auspices of the European Commission, the IDABC in March 2010 published the *Guideline on Public Procurement of Open Source Software*,[55] paragraph 2.4 of which recommended against using the term 'Open Source' but instead recommended using functional requirements 'which may include properties that are equivalent to the characteristics of Open Source software, or the characteristics of open standards'.

However, it is made clear that the *Guideline* is intended as a guide to good practice, and not as legal advice[56] and, in that context, one can understand why this recommendation is made.

The Public Sector Directive clarifies that award criteria must be detailed with the necessary transparency enabling all tenderers to be reasonably informed of the criteria.[57] It might be thought that the phrase 'Open Source' could be open to being differently understood by different potential tenderers and so lack the necessary transparency. Certainly a reference to functional characteristics might be thought preferable. Equally, if there were a concern over the use of the term, then the necessary transparency could equally be assured by referring to a recognised definition of that term, such as that provided by the OSI requiring the application of an approved licence.[58]

There has also been some discussion at the level of individual member states of whether or not specific reference to 'Open Source' can be made in a Specification. There has been little by way of guidance from the courts on this matter, though there are some useful indications in France, Italy, and Germany.

In Germany, the Directives are transcribed and public procurement regulated *inter alia* by *VGV Vergabeverordnung* (Procurement Order)[59] and *VOL/A Vergabe- und Vertragsordnung für Leistungen, Teil A* (Procurement Order applicable to services). Those rules are required to conform to the principle of the Wettbewerbsprinzip by virtue of section 97(1) GWB (Act Against Restraints of Competition). This principle requires that all applicants should be treated equally, and distinctions made only on the grounds of technical qualifications, efficiency, and reliability.[60]

[55] See note 37.

[56] Disclaimer at end of para 1.

[57] Recital 90 of the preamble, and see also Case C-496/99 *Commission v CAS Succhi di Frutta SpA* [2004] ECR 2004 I-03801 in relation to the 2004 Public Sector Directive.

[58] Open Source Initiative, <https://opensource.org/> accessed 8 February 2020.

[59] The paragraphs particularly applicable to Open Source are paras 4–7.

[60] Section 97(4).

Section 7 of *VOL/A* requires the specification in the tender documents for the procurement of software to be kept neutral, though section 7(3) does permit specification of Open Source software where reasonable, based on the type of the offer.

Interpreting these provisions, some writers have argued that specifying 'Open Source' would be in breach of the principle,[61] whereas other writers have suggested that, if the public body is able to explain why it is only Open Source which would fulfil their requirements, it would be permissible to specify Open Source.[62] It has been suggested that such reasons for the specification of Open Source may lie in higher security requirements which may indicate a need to alter the source code against 'back door' hacking, or a need for the contracting authority to acquire an Open Source licence together with the source code to allow it to develop or modify the software to meet its particular requirements.

In France, there is a similar legislative provision. The Directive is implemented by the *Code des Marchés Publics* (Code of Public Procurement Contracts). Article R211-7,[63] of the Code prohibits, *inter alia*, references to a trade mark, patents or type except where justified by the object of the contract or, exceptionally 'where a sufficiently precise and intelligible description of the object of the contract is not possible without it, and provided that it is accompanied by the words 'or equivalent'. Previously, the Conseil d'État had held that reference to a specific trade mark is permissible if it is the only means of describing the requirements of the contracting authority and provided that such reference is accompanied by the words 'or equivalent'.[64]

Apart from this general principle, no case in France has concerned a tender for the acquisition of Open Source as a species of good, though there is one case which deals with it in the context of the public procurement of services, a decision of the Conseil d'État dated 30 September 2011 regarding the Region of Picardie[65] (req no 350 43 Région Picardie). In that case, the Region of Picardie, as an acquiring authority, launched a procedure for the award of a public contract in order to 'carry out, exploit, maintain and host' an Open Source solution for the Digital Work Platform 'Lilie' (Espace Numérique de Travail (ENT)) for use in the colleges of the region. Two companies claimed that the Region failed to guarantee equal access and treatment to all tenderers contending that the technical specifications drawn up by the Region, by referring to an Open Source solution, had the effect of favouring certain undertakings.

[61] Dirk Heckmann, 'IT-Vergabe, Open source-Software and Vergaberecht' [2004] *Computer und Recht* 401, 408.

[62] 'Rechtliche Aspekte der Nutzung, Verbreitung und Weiterenwicklung von Open source-Software, November 2011'. Bundesbeauftragte fur Informatsionstechnik, Begleitdocument zum Migrationsleitfaden 4.0, s 30.

[63] As inserted by Décret no. 2018-1075 of 3 December 2018.

[64] Conseil d'État, 11 September 2006, Commune de Saran c Ste Gallaud, req no 257545.

[65] Req no. 350 43 Région Picardie.

Given the nature of the public procurement in question, the aim of which was not to supply software, but to *provide a service related to the software,* the Conseil d'État held that the mention of the Open Source solution in the technical specifications neither favoured nor penalised any potential tenderer. Indeed, the Conseil d'État pointed out that the Open Source being freely accessible, changeable, and adaptable by each company, to specify it in the technical specifications did not have the result of favouring any one tenderer over another.

It is important to emphasise that this case addresses the issue of public procurement *of services* (to carry out, exploit, maintain, and host the Open Source solution) and does not deal with the supplying of that Open Source solution (which may be regarded as public procurement of goods). The specific question of whether it is lawful for a contracting authority to specify Open Source in an exercise for the public procurement of goods remains a question which the French courts have yet to address.[66]

In Italy, in 2010, there was an important decision of the Constitutional Court regarding a regional law enacted by the Region of Piedmont[67] where the relevant regional law permitted contracting authorities, when assessing tenders, to give greater weight to those tenders that provided for the use of Open Source.

The then-applicable national law regarding public software procurement[68] required a contracting authority to choose amongst a number of specified options, one of which was Open Source. That choice was required to be made on the basis of a technical and commercial comparison,[69] but no further guidance was given in the national law as to how to carry out that comparison, so the general principles of procurement law applied.

The regional law had sought to give a boost to Open Source software by requiring the Region to use software applications of which the source code was available to it, and which it could freely modify to adapt the applications to its needs,[70] providing that, in the procurement of software, the Region should give preferential treatment to free software and software where the source code is accessible[71] and provided that if the Region were to choose proprietary software, it was required to justify the reasons for its choice.[72] Similarly a preference for Open Source was expressed in other provisions regarding data protection[73] and publicly accessible documents.[74]

[66] See *La Semaine Juridique Entreprises et Affaires no. 48*, 1 Décembre 2011, 1854.

[67] Decision no. 122 of 22/03/2010 available at <http//www.cortecosttituzionale.it/> accessed 18 January 2020. See also Carlo Piana, 'Italian Constitutional Court Gives Way to Free Software Friendly Laws' (2010) 2 *IFOSS Law Review* 61–66.

[68] 'Codice dell'Amministrazione Digitale' Dlgs no 82/2005. Article 68.

[69] *Assoli v Ministero del Lavoro* (TAR (Regional administrative court) Lazio, Decision no 428 of 23/01/2007.

[70] Art 6.1.

[71] Art 6.2.

[72] Art 6.4.

[73] Art 5.

[74] Art 4.

The national government challenged those provisions in terms of Article 117 of the Italian Constitution, as being in conflict with the rules of competition law laid down by the jurisprudence of the ECJ and the National Code of Public Contracts. The national government argued that the Region had to remain neutral in respect of the different technologies which might compete in a procurement exercise, that naming one particular technology over all others was clearly prohibited, and by extension, that giving preferential treatment to certain technologies, including or 'on the basis of' their licensing regime, should also be prohibited.

The court's analysis was that a requirement for open code is not a technical requirement but refers to a legal characteristic. It is open to contracting authorities to specify which licensing regime they require.

As Piana argued,[75] the distinction was based upon not only the technical and economic merit of the tenders but a non-technical characteristic, namely the nature of the legal rights offered.

In 2012, Article 68 of the Codice dell' Amministrazione Digitale[76] was amended to permit the use of Proprietary Software only where the comparative assessment demonstrates the impossibility of adopting Open Source software solutions or other solutions already developed in the public administration. This text was further amended later in 2012 by adding a sixth option (cloud computing) to the existing five, specifying in greater detail the criteria to be used in the comparative assessment,[77] and slightly rewording the text of the final sub-paragraph to clarify its meaning.

21.4.5.5 Excluding Open Source

The other side of the same coin is the issue of whether a contracting authority can specifically exclude Open Source solutions. Logically that may be so, consistently with *Concordia Bus*, provided that there is an objectively justifiable reason for doing so, though the issue that seems to be encountered more frequently is simply a specification which, in functional terms excludes Open Source.

Examples of functional exclusion can be found in *Elekta Ltd v Common Services Agency*[78] and the *SAS Anywhere Services* case.[79]

Interestingly, the debate in Poland regarding Open Source in procurement specifications seems to have been conducted in terms of the explicit exclusion of Open Source, rather than a positive requirement for Open Source. The principle of equal treatment is found not only in the Treaties, but is enshrined in national law in Article 32 of the Polish Constitution, which provides: 'Everyone is equal before the law. Everyone has the right to equal treatment by public authorities. No one

[75] Piana, see note 67.
[76] See note 68.
[77] Including interoperability and use of open standards.
[78] See note 49.
[79] See note 54.

shall be discriminated against in political, social or economic life for any reason.' According to the Constitutional Court, this means that persons who can be characterised by the same important feature are required to be treated equally without any adverse or favourable discrimination.[80]

Public authorities carrying out procurement exercises are bound both by this over-riding constitutional requirement, as well also as by the Polish laws on public procurement.

Public procurement in Poland was previously governed by the Law of 29 January 2004 on Public Procurement,[81] but the new regime, under the Law of 11 September 2019 on Public Procurement,[82] entered into force on 1 January 2021.

Under Article 7 of the 2004 Law, public authorities were obliged to prepare and conduct public procurement proceedings in a manner which ensured fair competition and equal treatment of suppliers. All the activities related to the preparation and carrying out of a procurement exercise were required to be conducted impartially and objectively. Article 29(2) contained a general prohibition on framing the tender specification, in such a way as would hinder fair competition, and Article 29(3) prohibited describing the object of the contract by reference to trade marks, patents, or origin, unless such description is justified by (i) the specific nature of the contract, and (ii) the inability of the public authority is to describe sufficiently the object of the tender by means of precise terms. Further, where such a reference was made, the description must be followed by the phrase 'or equivalent'.

The net result[83] was that all requirements had to be justified by the real and objective needs of the acquiring authorities. Only such limitation as was justified by real needs did not violate the law.[84] The formulation of mandatory requirements without reasons was forbidden and considered as discrimination.[85] The new regime makes similar provision.

In commenting upon this, Krzysztof Siewicz (a leading Polish legal writer) commented that the effect of this is that a public authority may not exclude or refuse to choose software just because it is Open Source software.[86] Any requirements which lead to the exclusion of Open Source software are required to be objectively justified. He suggests that a formulation of requirements which excludes providers of free software, would seem not to be objectively justifiable since, as he explains, Open Source software differs from closed or proprietary software because of the wider range of users' rights, but there is no objective reason why it could not

[80] Judgment of Constitutional Court of 9 March 1988, file no: U 7/87, OTK 1988, no 1, poz 1, 14.

[81] Dz U z 2019, r, poz 1843.

[82] Dz U z 2019 poz 2019.

[83] Consistently with the position in EU law.

[84] Decision of Krajowa Izba Odwoławcza of 5 August 2009 (file no: KIO/UZP 961/09).

[85] Decision of Krajowa Izba Odwoławcza of 13 January 2009 (file no: KIO/UZP 1502/08).

[86] Krzysztof Siewicz, *Prawne aspekty zamowień publicznych na oprogramowanie* (Poznań: Fundacja Wolnego i Otwartego Oprogramowania, 2010) 15, previously available at <http://www.fwioo.pl/media/attachments/prawne_aspekty_zamowien_publicznych_na_oprogramowanie.pdf> accessed 21 July 2022.

have the same or similar functionality. According to Siewicz, this means that the public authority may not refuse to choose software just because it is Open Source software.[87]

21.4.5.6 Summary

What then are the conclusions which can be drawn from all of this?

The application of the Treaties and derived principles is, according to the jurisprudence of the ECJ, quite clear: a contracting authority may generally not express a requirement or a preference for a particular named product or system. That prevents a specification which names UNIX, Intel, or Microsoft. However, that does not prevent its use with the addition of the magic words 'or equivalent'. But if there is no real equivalent, the words 'or equivalent' are, in reality, writ in water. This amounts to much the same thing as a specification which, functionally, closes down competition because it can be met by only one tenderer, or, at any rate, a very limited number of tenderers.

That situation is governed by the principles articulated in *Concordia Bus.* Such a closed or exclusive specification may be permissible, provided that the contracting authority can articulate reasons for drawing up the specification in those terms. But merely having reasons, which seem to the contracting authority to be good ones, is not enough. The absolute requirement, as the Advocate General makes clear in *Concordia Bus*, is that the reasons should be *objectively justifiable.* Note the combination of words: the focus is on justifiability and, what is more, objective justifiability.

These objectively justifiable reasons may relate to technical characteristics or non-technical characteristics. The *Piedmont* case appears to have been founded upon the proposition that Open Source software was preferred for its non-technical characteristics, but it is worth bearing in mind the peculiar circumstances in which the argument came to be based in Italian constitutional and administrative law, drawing on competition law, albeit that the context was acknowledged as involving procurement law principles. However, if one looks at the ECJ jurisprudence, ignoring the special Italian context, it is difficult to see why, in theory, a decision to specify Open Source software or to exclude Open Source software could not be justified on technical as well as non-technical grounds. Both would be equally permissible, provided that the contracting authority's reasons were objectively justifiable.

Similarly, the distinction drawn in the *Picardie* case between procurement of goods and procurement of services may not be of importance, for a decision to prefer or exclude Open Source solutions might equally be justified whether the procurement exercise relates to goods or to services, though it might be that the actual reasons may differ depending on whether the procurement exercise was

[87] Siewicz, see note 86.

characterised as one for goods or for services. It may be that, in the *Picardie* case, the Conseil d'État was mindful of the provisions of Article 6, IV of the *Code des Marchés Publics* with its references to 'products' and was reluctant to engage with whether Open Source software was a product, especially as the case could be (and was) decided without having reference to that issue.

It seems reasonably clear that 'Open Source' is not a specific product or brand, such as would bring into play the *Netherlands* and *Bent Mousten Vestergaard* principles. Accordingly, it is difficult to understand why there is academic disagreement in Germany on this point.

What it comes back to is that the seemingly absolute rules suffer exception where the contracting authority's reasons for departing from them are objectively justifiable, and one may be slow to state, categorically, as Siewicz does, that the exclusion of Open Source can never be justified (any more than one can assert that stipulating for it can always be justified). In any given case, what matters is the objective justifiability of the reasons.

It is for this reason that one might part company with the comments of Lord Glennie in *Elekta* if, in saying that a contracting authority 'cannot, in the interests of equal treatment, be compelled to seek tenders for something it does not want', he intended to lay down a rule of general application. Not only is the object of the tender something which the contracting authority wants, and must have its reasons for wanting, but also those reasons must be objectively justifiable.

21.4.5.7 Arguing objective justifiability

It may be that, in any given case, the ingenuity of the litigants will be able to produce plausible reasons why a decision by a contracting authority as to its requirements was or was not objectively justifiable. Such arguments may involve the economics of Open Source as against proprietary software or may involve technical issues or non-technical issues such as the licensing regime. That said, in the extensive jurisprudence and the policy thrust of the directives, and especially the 2014 Public Sector Directive with its focus on encouraging innovation, there is a wide range of potential arguments that might be used to assist those who wish to justify a selection (or attack the exclusion) of Open Source.

First, there is the economic argument. The underlying criticism of the reasoning of NHS Scotland in the *Elekta* case is that its decision to remain locked in was based on headline cost, and took no account of the ongoing cost of perpetuating (indeed, extending) lock-in. That this is not objectively justifiable is arguable from first principles, but it is in any event made explicit in the 2014 Public Sector Directive by the inclusion of lifecycle costing in Article 67(2) as an element in ascertaining the most economically advantageous tender read together with Article 68 setting out in detail what falls to be considered in a lifecycle costing exercise. Further, the UK Government Paper, 'Open Standards Principles: For software interoperability, data and document formats in government IT specifications'[88] states, at page 14: 'Short-term financial savings based only on cost could risk longer-term lock-in and are not necessarily the most cost-effective in terms of whole-life or when broader cross-government working or re-use is considered' and at pages 15 and 16:

> As part of examining the total cost of ownership of a government IT solution, the costs of exit for a component should be estimated at the start of implementation. As unlocking costs are identified, these must be associated with the incumbent supplier/system and not be associated with cost of new IT projects.

The paper recommends that in those exceptional cases where extensions to legacy solutions have been agreed, there should be formulated a 'pragmatic exit management strategy'. Two important warning notes should be sounded. These comments are made in the context of a policy on open standards and they are intended to bind only the UK Government and not the devolved national governments, local government or other public bodies. Whatever the context, the comments clearly embrace procurement policy; and, more importantly, the reasoning is compelling and indeed is echoed in the 2014 Directive. If the rationale of these comments is borne in mind, it is extremely difficult to see how anyone could have thought that, on economic grounds alone, the contracting authority's decision in *Elekta* could have been objectively justifiable.

So far as the technical and non-technical policy arguments are concerned, Article 67(2) of the 2014 Directive explicitly states:

> The most economically advantageous tender from the point of view of the contracting authority shall be identified on the basis of the price or cost, using a cost-effectiveness approach, such as life-cycle costing in accordance with Article 68, and may include the best price-quality ratio, which shall be assessed on the basis of criteria, including qualitative, environmental and/or social aspects, linked to the subject-matter of the public contract in question. Such criteria may comprise, for instance:
>
>> quality, including technical merit, aesthetic and functional characteristics, accessibility, design for all users, social, environmental and innovative characteristics and trading and its conditions . . . .

[88] HM Government, 'Open Standards Principles' (2012) <https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/78892/Open Standards-Principles-FINAL.pdf> accessed 30 January 2020.

This new, and innovation-friendly regime gives plenty of scope to justify a requirement for Open Source software and, consequently, more scope for challenging as not being objectively justified a decision to exclude Open Source, whether specifically or by way of an unduly restrictive specification. Such a legislative environment provides fertile soil for the further adoption of Open Source software.

21.4.6 The role of policy

A legal structure may be influenced by and reflect underlying policies, as with Article 67(2) of the 2014 Directive, but essentially it is a neutral framework providing no more than a mechanism for the implementation of policies.

At the EU level, a foundational document in the development of an Open Source-friendly policy environment was the *European Interoperability Framework Communication* of 23 March, 2017[89] which called for the removal of barriers to a digital single market in the EU. This was followed by the *Tallin Declaration on e-Government* of 6 October 2017,[90] in which the EU Member States unanimously called upon the Commission: 'to consider strengthening the requirements for the use of open source solutions and standards when (re)building of ICT systems and solutions takes place with EU funding, including by an appropriate open licence policy—by 2020'. Further impetus was given by the EU Parliament's July 2020 Briefing Paper on *Digital Sovereignty for Europe*.[91] The avowed aim of the paper is the development of 'a new policy approach designed to enhance Europe's strategic autonomy in the digital field'. There are clear signals in the text that as part of that vision, the European Parliament sees the importance of open solutions.

Unlike the older UK policies discussed later in this chapter, the Briefing Paper recognises the transition from software as a product to the platform or cloud economy as the relevant model of service delivery. It sees the need to adopt:

> a forward looking approach to digital markets' regulation and *to make online platform eco-systems and online activities more open*, fair and predictable. In particular, rules imposing algorithm transparency and neutrality, and data-sharing and interoperability could be considered. Finally, in a long-term perspective, fostering policies to build digital tools and solutions (e.g. operating systems and mobile platforms*) that avoid technology lockins and foster open*, yet still secure, digital ecosystems in the EU could be explored.[92]

[89] European Interoperability Framework–Implementation Strategy (COM(2017) 134).

[90] <https://ec.europa.eu/digital-single-market/en/news/ministerial-declaration-egovernment-tallinn-declaration> accessed 19 April 2022.

[91] <https://www.europarl.europa.eu/RegData/etudes/BRIE/2020/651992/EPRS_BRI(2020)651992_EN.pdf> accessed 19 April 2022.

[92] *Digital Sovereignty for Europe*, see note 91, page 8.

It lists a number of possible initiatives, including, as Initiative 1, the creation of an EU-wide cloud and data infrastructure.[93] Initiative 23[94] ('Foster open digital ecosystems') calls on the EU Institutions to '[a]ssess whether the EU framework should promote digital tools and solutions (e.g. operating systems) that avoid technology lock-ins'. It is noteworthy that initiative 1 is already being addressed in that the proposed EU Cloud is being built on Open Source software through the Gaia X Project.

Then, in October 2020, the EU Commission published its *Open Source Software Strategy 2020–2023—Think Open*.[95]

The Introduction sets out a clear-eyed vision of the importance of Open Source. It speaks of a desire: 'to bring Europe's people together in an inclusive, open approach, to find new opportunities and transition to an inclusive, better digital environment that is ready for the realities of today's global economy. In all of this, open source software has a role to play'. The document recognises the pervasiveness of Open Source software which is frequently missed by business and public sector users alike:

> "Open source is present everywhere. All around the world, companies and public services are using the open source collaborative methods to innovate and build new solutions. It powers the cloud and provides professional tools for big data and for information and knowledge management. It is in supercomputers, blockchain, the internet of things and artificial intelligence. It is in the internet. It is in our phones and our TVs. It provides us with streaming media. It is in our cars. It runs Europe's air traffic control. The chances are that, in any new project involving software, from kitchen appliances, through web-based public services to highly specialised industrial tools, most of the code will be based on open source
>
> and '[t]he 2020–2023 open source strategy reinforces the Commission's internal working culture that is already largely based on the principles of open source. However, it will also help the Commission change some of its technological and information management processes.'
>
> Section 2, *Vision*, states: 'The Commission leverages the transformative, innovative, and collaborative power of open source, encouraging the sharing and reuse of software solutions, knowledge and expertise, to deliver better European services that enrich society and focus on lowering costs to that society.' The governing principles of the policy are stated to be:[96]

[93] Initiative 1, page 9.

[94] Page 10.

[95] <https://ec.europa.eu/info/sites/info/files/en_ec_open_source_strategy_2020-2023.pdf> accessed 19 April 2022.

[96] Part 5, page 8.

Think Open - Open-source solutions will be preferred when equivalent in functionalities, total cost and cybersecurity

Transform—We harness the working principles of open source; we innovate and co-create, share and reuse, and together build user-centric, data-driven public services

Share—We share our code and enable incidental contributions to related open-source projects.

Contribute - We strive to be an active member of the diverse open-source ecosystem.

Secure—We make sure the code we use and the code we share is free from vulnerabilities by applying continuous security testing.

Stay in control—We promote open standards and specifications that are implemented and distributed in open source.

The Commission has also set up an OSPO, and is calling for these to be built across member states, taking the learning of the private sector OSPO discussed in Chapter 19 to the public sector.

These developments indicate a clear understanding of Open Source as a critical element in delivering on EU digital market policy commitments and provide the policy impetus required to encourage contracting authorities both at EU level and in the member states to use the legal structures creatively and in a manner delivering on the policy commitment to Open Source.

21.5 The UK

21.5.1 The legal framework

At the time of writing, EU law remains applicable to the UK as retained EU law, even although the UK ended its membership of the European Union at 11.00 p.m. GMT on 31 January 2020 and the transition period came to an end on 31 December 2020 with the Withdrawal Treaty. The existing rules will continue to apply unless amended in the future. The EU procurement regime comprises both the Treaty principles and the relevant EU Directives which are transposed into English and Scots law by, respectively, the Public Contracts Regulations 2015[97] and the Public Contracts (Scotland) Regulations 2015,[98] and so the rules contained within the Directives will also continue to apply as UK domestic law.

The UK government paper 'Public-sector procurement under the EU Withdrawal Agreement—Information for public authorities, businesses and other

[97] SI 2015/102.

[98] SI 2015/446.

organisations on the outcomes for public procurement after 31 January 2020'[99] confirms that under the Agreement, 'EU procurement law will continue to apply beyond the end of the transition period and for procurement procedures ongoing at the end of the transition period, the existing regulatory regime will continue up until award'.

The UK is a member of the WTO Agreement on Government Procurement (GPA)[100] through its former EU membership and intends to apply for membership of the GPA in its own right post Brexit.[101] The GPA covers government and sub-government entities as listed and each signatory may specify which entities it wishes to make subject to the regime and which thresholds to apply. An up-to-date list is maintained on the WTO website.[102]

21.5.2 The policy context

21.5.2.1 Organisational structure:

The UK government has been widely regarded as being at the leading edge in its policies on Open Source software since around 2012, when an industry-focused Cabinet Office Committee advised the government on the development of an Open Source strategy. The UK was also a signatory to the Tallinn Declaration in 2017 and has a consistent record of seeking to promote interoperability and the creation of a level playing field in the procurement of Open Source.

Although the UK does not have any legislation requiring the use of or governing the use of Open Source software, it does have a Technology Code of Practice[103] which requires public administrations to 'be open and use open'.

At UK government level, the primary responsibility for Open Source lies with the Government Digital Service (GDS), a part of the Cabinet Office. Within the GDS, a new Central Digital and Data Office for Government was set up in 2021 and a GDS Chief Digital Officer role created, reporting into the Cabinet Office. Amongst the new roles will be those supporting Crown Commercial Services on reforming technology procurement.

The Cabinet Office team is in control of the overall user experience across all digital channels and the team reports to the Secretary of State for Culture, Media and Sport (DCMS). Its responsibilities include: setting and enforcement of standards for digital services; building platforms and services; supporting the use of emerging technologies in the public sector; and leading the Digital, Data and

[99] European Union Withdrawal Act section 2.
[100] See note 4.
[101] Department for International Trade, Preparing for our future UK trade policy, 9 October 2017.
[102] <https://e-gpa.wto.org/en/Agreement/Latest> accessed 18 February 2020.
[103] <https://www.gov.uk/government/publications/technology-code-of-practice/technology-code-of-practice> accessed 12 April 2021.

Technology function for the government. Although the responsibilities do not include any specific responsibility for Open Source software, given the policy context, this can be regarded as being implicit in its remit.

The Crown Commercial Service (CCS) is an executive agency also sponsored by the Cabinet Office. It manages and implements the UK government's Open Source software policy 'GOV.UK' and has responsibility for its procurement.

21.5.2.2 Development of UK government policy

21.5.2.2.1 Open Source strategy for government

The UK government published this strategy document setting out the actions which needed to be taken by government to provide a level playing field between Open Source and proprietary software in the procurement process.[104] The opening lines read: 'This business plan has been developed as means of establishing how government will address its commitment to creating a level playing field for open source software.' The strategy's key objectives included 'strengthening the skills of public servants and suppliers in open source; and ensuring that open source, sharing, reuse and collaborative development are embedded in the culture of the government'. Although a resounding commitment to Open Source, there had not been an update to this strategy produced since 2010 until the 2022 Digital and Technology Playbook, which follows similar principles.

21.5.2.2.2 Advisory groups' business plan

This business plan, which was published in 2010, remains publicly available.[105] When it was published, it was world-leading and has been replicated in a number of other countries. Although the plan needs to be updated to take account of the considerable changes in the digital market coming from the transition to platform and cloud-based services, its principles remain relevant. Indeed, the work currently being undertaken by the EU institutions is fully in line with those principles.

21.5.2.2.3 The Government ICT Strategy

This Strategy document, issued by the Cabinet Office in 2011, aims to ensure the creation of a level playing field for Open Source software and the procurement of Open Source solutions.[106] The government established various groups that aimed

[104] <https://www.gov.uk/government/publications/Open Source-open standards-and-re-use-government-action-plan> accessed 19 April 2022.

[105] <https://www.whatdotheyknow.com/request/72114/response/189441/attach/3/Annex%20A%20Open%20Source%20Strategy%20Overview.pdf?cookie_passthrough=1> accessed 12 April 2021.

[106] <https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/85968/uk-government-government-ict-strategy_0.pdf> accessed 12 April 2021.

'to educate, promote and facilitate the technical and cultural change needed to increase the use of open source across government', including the Open Source Implementation Group, the System Integrator Forum, and the Government Open Source Advisory Panel, although the only group which remains active today is the Cabinet Office Open Standards Board.

21.5.2.2.4 Procurement Policy Note 8/11—Procurement of open Source IT

This Policy Note was published in 2011 jointly by the Cabinet Office, Efficiency and Reform Group and the CCS in order to encourage the use of Open Source by government departments. It states that when purchasing ICT solutions, 'government departments should ensure that open source software is fairly considered'.[107]

21.5.2.2.5 The Open Standards Principles policy document

The Open Standards Principles policy document[108] published by the UK government in 2012 aims to promote the use of open standards by public administrations. The policy document outlines principles for the selection and specification of open standards that can be implemented in both open source and proprietary software solutions.

21.5.2.2.6 The G-Cloud procurement system

Set up by the UK government in 2012, this system is used as a mechanism for the conducting of public procurements. This system itself contains a number of digital services and inevitably some of these are or include Open Source software. G-Cloud 12, its latest iteration, was released in early 2020.

21.5.2.2.7 The Government Transformation Strategy 2017–2020

The publication of this strategy in 2017[109] was an important development. The objectives of the strategy include: 'to create, operate, iterate and embed good use of shared platforms and reusable business capabilities to speed up transformation—including shared patterns, components and establishing open standards'. Although open standards are referred to, Open Source is not. However, the strategy does contain the following commitment: 'Building on the Digital Marketplace's approach, we will embed user-centred, design-led, data-driven and open approaches in procurement and contracting across government by 2020.'

[107] <https://www.gov.uk/government/publications/procurement-policy-note-8-11-procurement-of-Open Source> accessed 19 April 2022.

[108] <https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/78892/Open-Standards-Principles-FINAL.pdf> accessed 19 April 2022.

[109] <https://www.gov.uk/government/publications/government-transformation-strategy-2017-to-2020/government-transformation-strategy> accessed 19 April 2022.

21.5.2.2.8 Technology Code of Practice

This Code,[110] which remains current as of April 2021, includes a specific recommendation,[111] 'Be Open and Use Open Source', stating: 'Publish your code and use open source to improve transparency, flexibility and accountability', and detailed guidance is given in support of that recommendation.

21.5.2.2.9 The Local Digital Declaration

This is a joint initiative by the UK Ministry for Housing, Communities and Local Government, GDS, and a collection of local authorities and sector bodies from across the UK to encourage public authorities below the level of the UK government to improve IT procurement practices.[112] It does not lay down binding policies but takes the form of a Declaration which public authorities are invited to sign. The Declaration refers to open standards but not specifically to Open Source software.

21.5.2.3 Recent developments at UK level

The above policy documents appear to show willingness, as far as the implementation of forward-looking IT procurement practices is concerned. However, as in fact implemented, the policies did not have teeth and did not lead to widespread adoption of Open Source across the public sector until recently. This is encouraging, but because the policy landscape is fragmented, made up of a patchwork of policies some of which are a decade old, it is arguable that these policies are ripe for review at the same time as the current review of the National Data and Digital Strategies which is now under way.

DCMS produced a Data Strategy Consultation in 2019, and has announced that the Data Strategy will be issued in parts through the course of the year. The responsible Minister has confirmed that this will be updated from the existing strategies. However, at the time of writing the finalised strategy document is awaited and ten priorities have been issued by Oliver Dowden, Minister DCMS, none of which include or recognise opens-source software.

A further development is the proposed appointment by the Cabinet Office of a new publicly appointed Open Standards Advisory Board. In 2021 the Cabinet Office announced that it was seeking to appoint a group of ten advisors for the Standards field and rolled Open Source software into the work of this group. The announcement also recognised the importance of standards to technology development, and it is likely that advice on Open Source software will form an important subset of the work of this Board.

Policy initiatives are important, but of equal importance is the government being seen to lead by example. In this regard, amongst a number of notable recent

[110] <https://www.gov.uk/government/publications/government-transformation-strategy-2017-to-2020/government-transformation-strategy> accessed 19 April 2022.

[111] Point 3—see <https://www.gov.uk/guidance/be-open-and-use-open-source

[112] <https://localdigital.gov.uk/declaration/> accessed 19 April 2022.

adoptions of Open Source within the UK public sector has been the COVID-19 Test and Trace app created by NHSX, the digital arm of the NHS. NHSX develops code on a policy of open sourcing all code developed by it with its preferred licence being the MPL (Mozilla Public Licence). The code developed by NHSX, including the test and trace app, is not developed in the open but shared on completion via an Open Source software repository in GitHub (a popular public repository for the sharing of code with an Open Source licence).

The UK government has been criticised for this approach, being compared unfavourably to other governments such as the German federal government which developed its code in the open as well as licensing it as Open Source software. Because it was able to take community contributions during this development process, this significantly improved the outputs. Nonetheless, this does demonstrate a real investment by the UK government in the open solutions.

21.5.3 Devolved governments

The devolved government in Scotland has its own procurement regime, which builds on the EU regime. The UK government hosts a GitHub repository of its Open Source content and contribution guidelines.

The Scottish government has launched several Open Source initiatives, including its own NHSX COVID-19 app which is based upon the app provided by the Irish commercial vendor Nearform to the government of the Irish Republic. The base app was donated to the Linux Foundation, which is now the app's maintainer. The Scottish, Northern Irish, and Irish governments each have their own iteration of this app, with the potential for the respective apps to work cross border.

21.5.4 The third sector

21.5.4.1 OpenUK

A third-sector organisation, OpenUK, has been identified by the European Commission as the main actor in the UK in advocacy around Open Source. Its Legal and Policy group hosts a Future Leaders Group which has published a review of Open Source procurement by CCS.[113]

21.5.4.2 The State of Open

Phase one of the OpenUK report, *The State of Open*, was published in March 2021.[114]

[113] <http://www.Openuk.uk/stateofopen> accessed 19 April 2022.

[114] <https://openuk.uk/wp-content/uploads/2021/07/State-of-Open-Phase-Two.pdf> accessed 30 June 2022.

The report discusses the contribution of Open Source to the UK economy. The most recent verifiable figures showed 126,000 developers actively contributing to open source software projects in the UK with up to £43.15 billion contributed to the UK economy each year through open source working, though these figures were likely to be out of date. Pre-Brexit, the UK was the EU's largest Open Source contributor, considerably ahead of Germany and France. As a consequence of Brexit, the EU Commission, in its 2021 Report,[115] reduced the reported number of open source developers in the EU by 230,000 (though it considered that reduction to be overly conservative). This source therefore suggests that the true number of open source developers in the UK today is likely to be nearer 200,000 and the value to the UK economy significantly higher.

As the OpenUK report discloses, the UK is the fifth largest contributor to the Cloud Native Environment, one of the world's largest contributors to Open Source, and has one of the world's largest numbers of users. Many of the users and contributors to Open Source are not involved in avowedly Open Source companies but are working in mainstream business, in other technology companies, or working remotely for international companies, or across the public sector.

21.5.4.3 Conclusion

From the above, we see that the UK is a major player in the field of Open Source and other open technologies. This is reflected in the public sector where there has been a direct government commitment to open technologies and software since as long ago as 2010. Though some of the procurement policies which were developed are now becoming out of date, and not fully suited to the changed economic environment in which cloud has gained an increasingly dominant position, a review of these policies is now underway. When completed, it is likely to contribute to the UK's retaining its strong position. In short, in the field of public procurement of IT, the future for Open Source looks bright in the UK public sector.

21.6 The US

21.6.1 Introduction

The government sector in the US includes an immense federal government—by many counts the largest single procurer of goods in the world[116]—over fifty state,

[115] 'The impact of open source software and hardware on technological independence, competitiveness, and innovation in the EU economy', see <https://digital-strategy.ec.europa.eu/en/library/study-about-impact-open-source-software-and-hardware-technological-independence-competitiveness-and> accessed 30 June 2022.

[116] Brett Bachman, 'The US government is the world's largest purchaser of consumer goods. Amazon wants a piece' *Vox.com* (May 1, 2019) <https://www.vox.com/the-goods/2019/5/1/18524111/amazon-business-government-purchasing-state-city-local> accessed 19 April 2022.

tribal and territorial governments, and thousands of different county and municipality governments. Each of these entities is a potential procurer of software—including Open Source—and each generally has some degree of autonomy in setting the rules and regulations which govern the criteria, preferences, and procedures for software procurement. It is beyond the scope of this chapter to detail the many different policies governing procurement of software for all of these entities , but the various policies which have from time to time been published by the federal government and the larger state governments serve to give a good illustration of the state of public procurement in the US.

However, because policies have been published does not necessarily mean that smaller public entities in the US are familiar with them and follow them. Furthermore, in many cases, there may not exist specific policies concerning procurement of proprietary software as against Open Source, making it difficult to discern whether a given entity may favour or disfavour the procurement of Open Source.

21.6.2 The US federal government

With regard to the US federal government, procurement is dictated by two significant—and voluminous—sets of procurement regulations: the Federal Acquisition Regulations (FARS),[117] (used for US government procurement outside the US), the Department of Defense (DoD), and the Defense Federal Acquisition Regulations (DFARS),[118] used for procurement by the DoD. The DoD is, within the vast public procurement entity that is the US federal government, by far the largest procurer.[119]

Recognising that procuring relatively low-cost, low-volume, or non-government-specific items, including software, might not be done effectively by dictating that the supplier follow all the guidance of FARS and DFARS, there is an exception in both regulations for 'Commercially available off-the-shelf' items (COTS), defined as any 'item of supply' that is either sold or offered to government in the same form in which it is sold.[120]

[117] 48 CFR, Chapter 1. This chapter of the federal Code of Regulations sets for general guidelines (exclusive of the DoD) for all federal agencies; chapters 3 *et seq.* outline specific procurement rules for individual federal agencies that may differ from the general FARS.

[118] 48 CFR, Chapter 2.

[119] For fiscal year 2007, the DoD represented roughly three-quarters of the procurement budget of the entire US federal government. Federal Procurement Data System—Next Generation 'Federal Procurement Report, FY 2007' <https://www.fpds.gov/downloads/FPR_Reports/fy_2007/Total%20Federal%20View.pdf> accessed 19 April 2022, at pages 17 and 18.

[120] 48 CFR, Chapter 1, § 2.101.

The FARS also contemplates the acquisition of 'commercial computer software', defined as 'any computer software that is a commercial item', where a 'commercial item' is defined as being sold, leased, licened, or offered as such to the general public. The federal government has made clear that Open Source, which is of course specifically designed to be licensed to the general public, qualifies as both as 'commercial computer software' and as a COTS under the FARS and DFARS regulations despite the lack of its being sold or offered for sale.[121] As such, it may be procured by the U.S. federal government outside the rigours of standard procurement procedures, much as any off-the-shelf consumer proprietary software may be so acquired.

In addition to standard questions of direct procurement of software that may be offered under an Open Source licence, the US government has launched a number of initiatives in an effort to make sure that a percentage of the code that it develops internally, or that it procures externally, will eventually be released under free and Open Source licence terms. In 2016, the Chief Information Officer and Chief Acquisition Officer of the US released a memorandum announcing a policy that established a pilot program requiring at least 20 per cent of new custom code to be developed as Open Source for a three-year period and for metrics to be gathered during that time.

The results of this pilot program have been summarised in a recent article, indicating a substantial increase in the use of Open Source and the hosting of Open Source projects by the federal government, and that 'most [US federal government] agencies see value in using and publishing Open Source, agencies seeking more guidance'.[122]

Assuming this trend continues, the US federal government may continue to advance as both a consumer and a creator of software released under free and Open Source licence terms.

It is also notable that at least one agency of the US federal government has its own OSI-approved Open Source licence: the National Aeronautics and Space Administration (NASA) Open Source Agreement.[123] This licence was drafted, at least in part, so as to address some issues said to be unique to copyright ownership and licensing with US federal government authored, or US federal government contracted-for, software. This licence has been criticised as containing ambiguities that may be hindering its use.[124] Efforts to revise the licence to address these and

[121] Chief Information Officer, US Department of Defense, 'DoD Open Source Software (OSS) FAQ' part 4.1<https://dodcio.defense.gov/Open-Source-Software-FAQ/#Q:_Is_open_source_software_commercial_software.3F_Is_it_COTS.3F> accessed 16 July 2020.

[122] Joseph Castle, 'Happy 3rd Birthday Code.gov!' (20 November 2019) <https://digital.gov/pdf/Happy3rdBDayCode.gov.pdf> accessed 19 April 2022.

[123] <https://opensource.org/licenses/NASA-1.3> accessed 19 April 2022.

[124] National Academies of Sciences, Engineering, and Medicine, 'Open Source Software Policy Options for NASA Earth and Space Sciences' § 2.4.2. (2018). <https://www.nap.edu/read/25217/chapter/4#23> accessed 19 April 2022.

other concerns (including concerns that because the US federal government may not own US copyrights,[125] the licence might be partially ineffective) were ultimately unsuccessful, and the revised licence was rejected for approval by the OSI.[126] Nevertheless, there are numerous examples of software released on <http://www.code.gov> (the US federal government's Source Code repository) under a variety of other OSI-approved Open Source licences, although the most common licences used seem to be from the non-OSI-approved Creative Commons licence family. NASA has also been notable in 2021 in its utilisation of Open Source in the Perseverance Rover landing on Mars and for including Open Source and in particular Linux in this.

21.6.3 US state governments

Spurred on in part by the actions of the US federal government, at least one state of the US has announced its own policies and portals for Open Source development and use. California, by far the largest state in terms of population in the US, launched its own policies in 2018;[127] the policies are directed primarily to ensuring that California state government agencies are sharing code between themselves, and do not appear to require licences that would generally be considered free and Open Source, or that this code should be made accessible to the public. The California government's GitHub repository[128] does include a handful of projects, including several that are licensed under OSI-approved open source licences.

Other large US state governments (Texas, Florida, and New York are the next largest states in the US by population) do not have clearly articulated policies favouring or encouraging Open Source use by those governments. Given that the policy announcements by the US federal government are relatively new, having been issued in 2016, the fact that many US states have not developed a coherent policy on Open Source, or code repositories where the public may have access to state government developed or contracted-for software, may be a symptom of US states adopting policies reactively rather than proactively. Even the City of San Francisco—arguably the centre of the tech world—lacks any sort of coherent policy or practice encouraging use and adoption of Open Source code.[129] The adoption of

[125] 17 USC § 105.

[126] Open Source Initiative, 'License Committee Report—January 2017' (9 January 2017) <https://lists.opensource.org/pipermail/license-review_lists.opensource.org/2017-January/002933.html> accessed 19 April 2022.

[127] 'Code California Playbook' <https://codecagov-playbook.readthedocs.io/en/latest/policy/> accessed 16 July 2020.

[128] <https://github.com/CA-CODE-Works> accessed 19 April 2022.

[129] Charles Belle, 'Open Source & The City: Making SFGov a Leader in Tech Policy' *Reset San Francisco* <https://www.resetsanfrancisco.org/better-government/Open Source-city-making-sfgov-leader-tech-policy/> accessed 16 July 2020.

a policy by the state of California may lead other states—or smaller governmental units like counties or municipalities—to model policies based on the leads given by the US federal government or California.

21.6.4 Conclusion

In summary then, despite the lead that private industry in the US has in developing, hosting, and sharing Open Source, it is surprising that governmental entities in the US lag behind their peers in other countries in setting official policies and practices to encourage these practices. The Open Source communities are optimistic in light of the change of President and the appointment of a Special Assistant to the President and Director of Technology, David Recordon, who has a background in Open Source and open standards.

21.7 Conclusion

Where then stands the relationship between public procurement and Open Source?

At the time of the first edition of this book, the business model largely involved the delivery of a software solution as such, and developers and suppliers could find themselves effectively, and often unfairly, shut out of procurement contracts by the creation of inappropriate contract specifications or the application of inappropriate award criteria. With now an increasing reliance on software as a service, that problem may be masked to some extent, because contracting authorities judge the service offer which is provided rather than the software itself. However, the underlying software can be essential to achieving and maintaining service levels and providing innovative and cost-effective solutions: the apparent issue may be masked but it still lurks there and, indeed, there are still plenty of procurement exercises which do explicitly concern themselves with software.

For Open Source developers and providers, the applicable procurement regime still matters more than ever.

The EU legal regime is robust and detailed, with relatively clear legislative provisions. It covers, by means of the Directives, high-value tenders and by means of the Treaty Principles (as interpreted by the courts) below-threshold procurement (at any rate, so long as there is a potential cross-border interest). This regime avowedly and explicitly seeks to encourage innovation, both by allowing wider considerations than merely price in the criteria by which tenders may fall to be judged, including an explicit commitment to lifecycle costing as part of those criteria and by the mechanism of innovation partnerships. The nature of the legal structures is broadly favourable to Open Source developers and suppliers who can make out a good commercial case for the use of Open Source—a case which members of the Open Source community are well able to put.

These comments apply equally to the market in the UK, and with its history of Open Source-friendly policies and ongoing policy reviews, the future for Open Source remains hopeful. Of course, there are doubts and uncertainties arising from Brexit. For example, will the UK actually succeed in becoming a member of the GPA in its own right? Which entities and thresholds will the UK government list in the Annexes? What opportunities will there continue to be for UK companies to participate in EU public procurements and for EU companies to participate in UK public procurement exercises? Assuming there is an equivalence to the EU procurement thresholds, small and medium-sized enterprises depend on getting a steady stream of below-threshold work, but what will be the opportunities for UK companies to participate in below threshold procurement exercises in the EU?

The WTO's GPA is certainly of some assistance in allowing UK participation in public procurement exercises in both the EU and elsewhere, but it falls far short of the robust in-depth regime provided by EU law.

By contrast with the comprehensive and detailed regime in Europe, public sector procurement in the US is governed by a patchwork of different regimes, only some of which are what could be described as 'open source friendly', and such rules as there are, are relatively underdeveloped. The present state of Open Source procurement in the public sector in the US compares unfavourably with both the private sector in the US and with the EU and UK where there is a well-developed legal structure to govern procurement decisions.

The discussion, however, is not merely about legal structures. You can use legal structures to lead a horse to water, but you cannot make it drink: there has to be a will on the part of public authorities not necessarily to specify Open Source as such so much as to specify criteria which do not shut out Open Source. Members of the Open Source community can be confident that, in a fair fight, the advantages of Open Source will see off the proprietary opposition. The trick is to ensure the fairness of that fight.

The legal rules creating the playing field are relatively clear, though given the practical experience, one might be forgiven for thinking that the law is more complex than it is. In particular, it is clear under the EU procurement regime that all procurement exercises where there is a cross-border interest, whether above or below threshold, have to conform to the Treaty and derived principles, including transparency and equal treatment. Those principles exclude the limiting of technical specifications to named proprietary software but permit the specification of Open Source in general, or a functional definition which permits or even favours Open Source. The key is that the requirements of the specification (whether favouring or excluding Open Source) have to pass the test of being objectively justified.

It is questionable how far, in many tendering exercises, objective justification for the choices made was really a consideration, or, at least, the justifications were being looked at too narrowly by the contracting authority, and it may be open to question whether wider issues such as vendor lock-in were considered.

In the final analysis, procurement law is about how contracting authorities go about contracting, but it is neutral as to the content of the outcome of the tender exercise, provided the process meets the requirements of, *inter alia*, transparency and equal treatment. Those requirements may go some way towards levelling the playing field for Open Source, but an essential element (especially for public authorities who may have got too comfortable with their existing proprietary solutions) is clear and coherent public policy guidance.[130]

Public procurement can be a valuable tool in levelling the playing field, but it would be a mistake to see it as a panacea. Also critical to ensuring proper competition is the requirement for interoperability discussed further in Chapter 17.[131] Without interoperability, proper competition is prevented: vendor lock-in can happen too easily.

Since the first edition of this book, the GPA has gathered more signatories; the EU (and UK) regime is now, more than ever, receptive to the broader concerns which Open Source is better equipped to meet than proprietary rivals may be, and the EU and UK policy climate looks promising.

[130] See, for example, 21.4.2 and 21.5.6 above.

[131]

PART 3

EVERYTHING OPEN

22
Blockchain and Open Source

Mark Radcliffe

Blockchain technology has been described as one of the most important technologies in the last five years. It has potential application to multiple industries and provides the opportunity to significantly improve existing business processes and enable new businesses and on occasion. The discussion of the use of 'Open Source' in blockchain requires an understanding of blockchain technology and the broader category of Distributed Ledger Technology (DLT). The Cambridge Centre for Alternative Financing (the Centre) in its 2018 'Distributed Ledger Technology Systems: A Conceptual Framework' (Cambridge Report)[1] notes:

> Distributed ledger technology (DLT) has established itself as an umbrella term to designate multi-party systems that operate in an environment with no central operator or authority, despite parties who may be unreliable or malicious ('adversarial environment'). Blockchain technology is often considered a specific subset of the broader DLT universe that uses a particular data structure consisting of a chain of hash-linked blocks of data.[2]

[1] Michel Rauchs, Andrew Glidden, Brian Gordon, et al. and Cambridge Centre for Alternative Financing, 'Distributed Ledger Technology Systems: A Conceptual Framework' August 2018) <https://www.jbs.cam.ac.uk/faculty-research/centres/alternative-finance/publications/distributed-ledger-technology-systems/> accessed 30 June 2022.

[2] Cambridge Centre for Alternative Financing, 'Distributed Ledger Technology Systems: A Conceptual Framework', n.1, 12.

Mark Radcliffe, *Blockchain and Open Source* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press. © Mark Radcliffe 2022. DOI: 10.1093/oso/9780198862345.003.0022

After reviewing how third parties have defined DLT and the limitations of those definitions, the Cambridge Report suggests the following definition which is very useful:

> A DLT system is a system of electronic records that
>
> i. enables a network of independent participants to establish a consensus around
> ii. the authoritative ordering of cryptographically-validated ('signed') transactions.
> iii. These records are made persistent by replicating the data across multiple nodes, and tamper-evident by linking them by cryptographic hashes.
> iv. The shared result of the reconciliation/consensus process—the 'ledger'—serves as the authoritative version for these records.[3]

As noted earlier, one of the major challenges of discussing the use of Open Source software in blockchain systems is understanding the technical details of the software stack. However, the terms used to describe DLT are ambiguous. The Cambridge Report describes this problem as follows: 'The DLT ecosystem is plagued with the use of incomplete and inconsistent definitions and a lack of standardised terminology, creating a needlessly complicated landscape for everyone from experienced policymakers and developers to individuals venturing into the field for the first time',[4] and

> [a]dding to the challenge is that on the one hand, definitions are sometimes too specific, technical and inaccessible to general audiences; while on the other hand, some are too simplistic and broad so that no meaningful difference to more traditional database architectures can be observed. Either way, a lack of common terminology has resulted in misconceptions and the widespread formation of unrealistic expectations as to what this technology can achieve.[5]

However, industry groups are trying to standardise or, at least, clarify these different terms such as the work on tokens found in the Token Technology Framework, initially developed by the Token Taxonomy Initiative and now being managed by the Interwork Alliance.

Given this reality, this summary has made a number of choices to avoid undue complexity: (i) the chapter will use the term 'blockchain' rather than DLT because it is more common and most of the software reviewed are blockchains; (ii) the

[3] Cambridge Centre, 'DLT Systems', see n. 1, 20.
[4] Cambridge Centre, 'DLT Systems', see note 1, 10.
[5] Cambridge Centre, 'DLT Systems', see note 1, 81.

chapter will use common terms such as node and block with the understanding that their meanings may vary among different blockchains; and (iii) these systems are so constantly evolving and this discussion is based on code audits by the Black Duck Audit team at Synopsys, Inc. in September 2020 (and in the case of Besu and Corda, January 2020) but are not publicly available.[6]

The Cambridge Report divides blockchain into three layers: (i) the protocol layer; (ii) the network layer; and (iii) the data layer.[7] This chapter will focus on the 'protocol layer' software such as Bitcoin, Ethereum, EOS, and Cordera. However, the software stack for blockchain systems which build on top of these protocol layers is very complex and varies based on the blockchain system. The protocol layers, such as Bitcoin and Ethereum, are sometimes referred to as 'Layer 1' or the 'base layer'. Some blockchain systems also have 'Layer 2' software such as the 'Lightening Network' for Bitcoin. Blockchain systems also host application software, with smart contacts, tokens, and 'DApps', all of which are software.

22.1 Blockchain Systems

This section will provide a more detailed discussion of blockchain technology. Blockchain can be summarised as a distributed ledger based on blocks on which transactions are recorded based on a consensus protocol; some blockchains, particularly public blockchains such as Ethereum and Bitcoin, are decentralised. The transaction ledger is maintained simultaneously across a network of unrelated computers or servers called 'nodes', like a spreadsheet that is duplicated thousands of times across a network of computers. The decentralisation of a blockchain network is not a simple matter of counting the number of nodes in the network because blockchain networks using the Proof of Work consensus protocol have found that 'miners' aggregate into 'mining pools' who work together to solve the cryptographic problems needed to get the right to publish the next block. The ledger contains a continuous and complete record (the 'chain') of all transactions performed which are grouped into blocks: a block is only added to the chain if the nodes, which are members in the blockchain system, reach consensus on the next 'valid' block to be added to the chain. A transaction can only be verified and form part of a candidate block if the nodes on the network needed by such a network's protocol confirm that the transaction is valid; the method of determining this issue is called the 'consensus' protocol.

A critical factor in the choice of consensus protocol for a blockchain system is whether the system is public ('permissionless') like the Internet or structured within a private group like an intranet ('permissioned'). 'Permissioned'

[6] These are private scans and are not available on the Internet.
[7] Cambridge Centre, 'DLT Systems', see note 1, 11.

blockchains permit only certain pre-approved participants to join the blockchain system. Consequently, these blockchains can use a much wider number of consensus protocols and the members can agree on rules that govern how the blocks are recorded.

The best-known consensus protocol is 'Proof of Work' which is used by the two most widely known 'permissionless' blockchains, Bitcoin and Ethereum. The Proof of Work consensus protocol requires that the 'miner' solve a computationally intense cryptographic puzzle to get the right to publish the next block. However, other types of consensus protocol exist. For example, the National Institute of Standard and Technology 'Blockchain Technology Overview' (NIST Overview) describes two other consensus protocols: Proof of Stake and Round Robin Consensus.[8] Blockchains can also change their consensus protocol: the Ethereum blockchain is moving to Proof of Stake as its new consensus protocol in 2002.

The 'block' is the fundamental unit of a blockchain system, and the data fields in a typical block are described by NIST Overview as follows:

- The block number, also known as block height
- The current block hash value
- The previous block hash value
- The Merkle tree root hash (defined later in this chapter)
- A timestamp
- The size of the block
- The nonce value, which is a number manipulated by the mining node to solve the hash puzzle that gives them the right to publish the block
- A list of transactions included within the block[9]

Once information is entered on the blockchain, it is extremely difficult to alter: a blockchain system lacks a centralised point of vulnerability for hackers to exploit and each block includes the previous block's 'hash' of all previous blocks, so any attempts to alter any transaction with the blockchain are easily detectable because every block in the chain is affected. However, blockchains can be changed as described shortly in the section on forking.

22.2 Protocols and Clients

One unusual aspect of some blockchain systems is the existence of 'separate' codebases which provide the same functionality. The rules of a blockchain system

[8] National Institute of Standards and Technology (NIST), US Department of Commerce, 'Blockchain Technology Overview', Draft NISTIR 8202 (January 2018), 29–30. <https://doi.org/10.6028/NIST.IR.8202> accessed 30 June 2022.

[9] NIST, 'Blockchain Overview', see note 8, 15–16.

are described as a 'protocol'; the codebase which implements the protocol is described as a 'client'. These codebases are developed and maintained by different parties and frequently use different licences. For example, Ethereum has many clients, including Hyperledger Besu, CPP, Go, Quorum, and Parity. This chapter will review the Open Source licences of three of these 'clients' for Ethereum.

22.3 Forking

Although forking is frequently described as a critical advantage of Open Source software, it rarely happens to established Open Source projects. However, forks are more frequent in blockchain networks and take on significant importance. Since blockchains rely on a 'chain' of blocks using the protocol layer software, an update to such software will have different effects based on the nature of the update. The NIST Report describes the effects as follows:

> A soft fork is a change to the technology that *will not* completely prevent users who do not adopt the change (e.g., an update to the latest version) from using the changed blockchain system. Since non-updated nodes will recognize the new blocks as valid, a soft fork can be backwards compatible, only requiring that a majority of nodes upgrade to enforce the new soft fork rules.[10]

An example of a soft fork of the Bitcoin blockchain was BIP 66 which dealt with signature validation.

> A hard fork is a change to the technology that *will* completely prevent users who do not adopt it from using the changed blockchain system. Under a hard fork, the blockchain protocol will change in a manner that requires users to either upgrade to stay with the developer's 'main fork' or to continue on the original path without the upgrades. Users on different hard forks cannot interact with one another. Any change to the block structure, such as the hashing algorithm choice, will require a hard fork.[11]

The best-known hard fork is the fork of the Ethereum blockchain to reverse problems in the smart contract which ran the Decentralized Autonomous Organization (DAO) in 2016. The DAO was designed to automate investments in projects for the Ethereum blockchain. The DAO raised about 12.7 million Ether which was worth about US$150 million at the time. The DAO was hacked, and the hacker diverted about $50 million in Ether, but it was temporarily 'locked up' due to the terms of

[10] NIST, 'Blockchain Overview', see note 8, 29.

[11] NIST, 'Blockchain Overview', see note 8, 29–30.

the smart contract, and the hacker could not get access to it for twenty-seven days. After considering a soft fork, certain developers proposed a hard fork which was implemented. However, the hard fork needed to be implemented by the node operators, and about 89 per cent of the node operators accepted the new code and implemented the hard fork. The Ethereum blockchain split into two chains: Ethereum and Ethereum Classic. The Ethereum blockchain was composed of the nodes that accepted the software upgrade to reverse the Ether investments in the DAO. The Ethereum Classic blockchain was composed of the nodes that did not accept the software upgrade to reverse the Ether investments in the DAO.

The philosophy of blockchain developers, with its focus on decentralisation and empowerment of communities, ensured that Open Source licences would be a natural fit for blockchain projects. Moreover, several of the early blockchain developers had participated in other Open Source projects. Early projects such as Bitcoin and Ethereum were informally organised. The early days of the Ethereum project were described in detail in the recent book, *The Infinite Machine: How an Army of Crypto-Hackers in Building the Next Internet with Ethereum*.[12] Yet, Open Source licensing has two major philosophies: permissive and copyleft. The licences implementing these philosophies have very different obligations. The developers of three of the early Ethereum clients selected copyleft licences. However, the decentralised nature of blockchain projects could raise challenges with the compliance obligations of copyleft licences.

The different elements of technology underpinning blockchain has been in existence for decades, including e-cash protocols from the 1980s and 1990s, but the combination of technologies is revolutionary. It was described in *Bitcoin: A Peer-to-Peer Electronic Cash System* ('Bitcoin Whitepaper')[13] sent to an email list by Satoshi Nakamoto (who is still unknown) in 2009. The Bitcoin Whitepaper put these technologies together and started the blockchain revolution. Further, the Bitcoin Whitepaper described a digital asset with no backing or 'intrinsic value' and without a centralised issuer. The Bitcoin Whitepaper also described the concept of 'distributed consensus' to validate the new blocks in the chain. Bitcoin was very successful as a store of value, but its limitations as a more general platform for distributed applications encouraged Vitalik Buterin to develop a more flexible alternative. During 2013 and 2014, he worked on designing the Ethereum protocol. A group of developers assisted to work out the technical details of the Ethereum blockchain as described in the Infinite Machine. The first client of the Ethereum blockchain was distributed in early 2015.

[12] Camilo Russo, The Infinite Machine: How an Army of Crypto-Hackers in Building the Next Internet with Ethereum (New York, NY: Harper Business, 2020).

[13] Satoshi Nakamoto, 'Bitcoin: A peer-to-peer electronic cash' (31 October 2008) <https://bitcoin.org/en/bitcoin-paper> accessed 10 December 2020.

22.4 Code Review

We have reviewed the codebases of two Bitcoin clients and four Ethereum clients as well as two new blockchain platforms, EOS and R3's Corda. EOS and Corda are both platforms developed by corporations.

The audits were performed by the Black Duck Audit team at Synopsys, Inc. which is one of the leading Open Source scanning companies. These audits involve expert analysis using sophisticated software tools built on Black Duck KnowledgeBase™ comprising the code from millions of Open Source projects. The Black Duck audit process is described on the Synopsys website.[14] As those who have used Black Duck services are aware, the scan is the first step in understanding the licences of a project and its compliance. Once the scan is complete, the actual interaction of the software files must be reviewed based on potentially conflicting obligations in their licences. For example, the General Public Licence version 2 (GPLv2) has obligations that conflict with the obligations of the Apache Software Licence version 2 (Apache), but this conflict in obligations is only relevant if the projects under these two licences interact in a way which would trigger the obligations in the licence. In this case, the Apache licensed project would need to interact with the GPLv2 licensed project in a manner which creates a 'derivative work' as defined in the GPLv2. The review of this interaction is very technical and fact intensive. The author has not been able to perform this level of review, so the notes on the scans are indicative of potential problems rather than confirmed problems. Black Duck reports provide the total number of files in the project and distinguishes between the files which are in Black Duck KnowledgeBase (which the author refers to as 'third-party open source software') and the files which are not in their database and, thus, are assumed to be original. The third-party Open Source software may be complete projects or 'snippets' from other projects. Black Duck reports also use three terms in describing potential conflicts: broach reach, narrow reach, and component. A 'broad reach' conflict 'refers to the fact that (i) the reach of the licence goes well beyond the Open Source code that the licence comes with or (ii) it's difficult to use the code in a way that avoids a conflict. It refers to licences which apply to the entire derivative work (such as GPLv2).[15] A 'narrow reach' conflict refers to '[l]icenses in this category also have a declared conflict. However, they typically have less reach or are easier to use appropriately to prevent a conflict.' The Common Development and Distribution Licence is an example of a licence in this category. Generally, licences in this category generally have a narrower scope for interaction with third-party files.[16] A 'component conflict' refers to a 'status with

[14] <https://www.synopsys.com/software-integrity/managed-services/Open Source-software-audit.html> accessed 10 December 2020.

[15] <https://www.synopsys.com/blogs/software-security/Open Source-license-risks/> accessed 10 December 2020.

[16] <https://www.synopsys.com/blogs/software-security/Open Source-license-risks/> accessed 10 December 2020.

two or more components have licences that conflict with one another but can be used safely by themselves in a commercial codebase'.[17] Such licences do not conflict with the 'declared' project licence but potentially conflict with the licences for other components.

The scans for the projects based on Java (HyperLedger Besu and R3's Corda) were limited to the code in the GitHub repository because these files are typically calling other .jar files for the dependencies, and .jar files are generally considered dynamically linked. Most Open Source lawyers believe that dynamic linking rarely triggers obligations under Open Source licence between linked projects. However, the codebases for both Bitcoin clients, the other three Ethereum clients, and EOS in Github were built using their dependencies, and the scan is based on the 'built' project.

Black Duck based on its audits of software for its clients estimated in 2021 that seventy five percent (75%) of the code in software programs used by corporations is Open Source software.

The Bitcoin clients have significant amount of third-party Open Source projects: 78 per cent and 83 per cent. Both clients are under permissive licences, the MIT licence and a variant of the MIT licence. The Bitcoin clients also have a significant number of potential conflicts: Bitcoin SV (19 per cent) and Bitcoin (16 per cent) which suggests the difficulty of managing Open Source compliance in decentralised projects.

The earliest three Ethereum clients also included very significant amounts of third-party Open Source projects with 77 per cent, 84 per cent, and 88 per cent. They were also offered under copyleft project licences which require management of contributions to ensure compliance with the copyleft project licences. They also have a significant number of potential conflicts with the CPP Client (25 per cent), Parity Client (9 per cent), and Go Client (14 per cent). On the other hand, the Ethereum client developed by Pegasys, Hyperledger Besu, has very low third-party Open Source projects at 1 per cent. It is licensed under the Apache Software Licence version 2 to make it more attractive to corporate users.

The two company run projects, Corda and EOS, have very different profiles. R3's Corda has only 5 per cent third-party Open Source projects and has only 9 per cent potential licence conflicts. Corda is under a permissive licence, Apache Software Licence version 2. EOS has 42 per cent third-party Open Source projects with 24 per cent potential licence conflicts. EOS is under the MIT licence.

[17] <https://www.synopsys.com/blogs/software-security/Open Source-license-risks/> accessed 10 December 2020.

22.5 Bitcoin Client Licence Analysis

Bitcoin is the first blockchain and is based on the Bitcoin Whitepaper by Satoshi Nakamoto which was published in 2008. The original Bitcoin client was released on 9 January 2009. This history of Bitcoin and its clients is complex and opaque. Bitcoin.org lists fifty-four Bitcoin clients. The chapter will review two of the more commonly used clients.

Bitcoin Core was initially named Bitcoin-Qt. It was the third Bitcoin client and developed by Wladimir van der Laan based on the original reference code by Satoshi Nakamoto. Bitcoin-Qt version 0.5.0 was released on 1 November 2011. Bitcoin-Qt introduced a front end that uses the Qt user interface toolkit.

Bitcoin SV (Satoshi's Vision) is a fork of Bitcoin Cash client. Bitcoin Cash was released on 1 August 2017 as part of the 'bitcoin scalability debate.' The hard fork of Bitcoin Cash resulted in two competing coins: Bitcoin ABC (Adjustable Blocksize Cap) and Bitcoin SV. The fork was due to a disagreement over how to best solve the problem of 'scalability' in the Bitcoin blockchain. Bitcoin SV was led by Craig Wright who claims to be Satoshi Nakamoto, the original developer of Bitcoin.

22.5.1 Bitcoin core client

The scan of the Bitcoin Core client is for Version 0.19.0.1. and was performed on 4 September 2020. The project licence is MIT. The project includes 83 per cent third-party Open Source software. The project has 30,390 files of which 25,194 files include third-party Open Source software. These third-party Open Source files have seventy-six different third-party Open Source components under twenty-nine different Open Source licences. Synopsys noted that 16 per cent of the components are licenced under licences which could potentially cause conflicts. Under the BlackDuck analysis, three files (within one component) have a 'broader reach' licence conflict and no files have a 'narrower reach' licence conflict. The Bitcoin Core client has 1,009 files within eleven components that have potential component conflicts. The licences for the components of the project include many traditional Open Source licences including many copyleft licences such as General Public Licence version 2 (or later) (GPLv2+), General Public Licence version 3 (or later) (GPLv3+), Affero General Public Licence version 3, LGPL version 2.1 (or later), and LGPL version 3 (or later), as well as many permissive licences. However, it includes some unusual licences such as Licence for AMD64 Patch by Mikhail Teterin which provides a licence under Berkeley Software Distribution (BSD) but prohibits 'use by owners of Che Guevarra paraphernalia' where possible.

22.5.2 Bitcoin SV client

The scan of the Bitcoin SV client is for Version 1.0.0 and was performed on 4 September 2020. The project licence is Open BSV Licence, a licence specially drafted for the project (and it has not been submitted for or approved by the Open Source Initiative (OSI) as an 'Open Source' licence). The original text of the Open BSV Licence is set forth below:

> The Open BSV License
>
> Copyright (c) 2019 The Bitcoin SV developers
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the 'Software'), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> 1—The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
> 2—The Software, and any software that is derived from the Software or parts thereof,
> can only be used on the Bitcoin SV blockchain. The Bitcoin SV blockchain is defined,
> for purposes of this license, as the Bitcoin blockchain containing block height #556767
> with this hash: 000000000000000001d956714215d96ffc00e0afda4cd0a96c96f8d802b1662b.
> THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.[18]

The Open BSV Licence is the MIT licence with the addition of the following sentence:

[18] <https://github.com/bitcoin-sv/bitcoin-sv/commit/e6474ba84db58d8adf01354a8c12931a9d7e8d3d> accessed 10 December 2020.

> 2—The Software, and any software that is derived from the Software or parts thereof,
> can only be used on the Bitcoin SV blockchain. The Bitcoin SV blockchain is defined,
> for purposes of this license, as the Bitcoin blockchain containing block height #556767
> with this hash: 000000000000000001d956714215d96ffc00e0afda4cd0a96c96f8d802b1662b.

This modification is meant to prevent forking. However, it is not consistent with the Open Source Definition (OSD) and thus would probably be rejected by the OSI if it was submitted to qualify as 'open source'. This provision would also be inconsistent with the terms of the GPLv2+ and GPLv3+ if the projects licensed under the GPLv2+ and GPLv3+ interact with the project licensed under the Open BSV Licence in a manner which creates a 'derivative work' or 'modification'.

The project includes 78 per cent third-party Open Source software. The project has 41,268 files of which 32,365 files include third-party Open Source software. These third-party Open Source files have seventy-nine different third-party Open Source components under twenty-seven different Open Source licences. BlackDuck noted that 19 per cent of the components are licensed under licences which could potentially cause conflicts. Under the BlackDuck analysis, three files (within one component) have a 'broader reach' licence conflict and no files have a 'narrower reach' licence conflict. The project also includes 7,291 files within fourteen components that have potential component conflicts. Similar to the Bitcoin Core project, the licences for the project include many traditional Open Source licences including many copyleft licences such as GPLv2+, GPLv3+, Affero General Public Licence version 3, LGPL version 2.1 (or later), and LGPL version 3 (or later), as well as many permissive licences. Once again, similar to Bitcoin Core project, it includes some unusual licences such as Licence for AMD64 Patch by Mikhail Teterin (just as in Bitcoin Core) which provides a licence under BSD but prohibits 'use by owners of Che Guevarra paraphernalia' where possible.

22.6 Ethereum Client Licence History

We are fortunate to have insight into the discussions concerning licensing of one of the first Ethereum clients, cpp-ethereum (sometimes referred to as the C++ client and now named 'aleth') ("CPP Client") through the assistance of one of the participants in these discussions. He worked as part of the team of developers on the CPP Client and tried to organise a relicensing of the project.

The CPP Client was developed primarily by Gavin Wood both before and during the meetings in Switzerland among the Ethereum founders about how to legally structure the Ethereum project. The discussion of the licences started in December 2013 with a wider range of proposals from public domain to permissive to copyleft:

> Thursday, 26 December 2013
> [16:29:14] vbuterins: also, for formality's sake, we should nail down the *licenses for all of our open* source stuff
> [16:29:27] vbuterins: I'll make mine public domain
> [16:31:20] Charles Hoskinson: BSD, Apache, MIT
> [16:31:33] Charles Hoskinson: You could also use GNU
> [16:34:45] Charles Hoskinson: <http://opensource.org/licenses>
> Friday, 27 December 2013
> [09:37:17] gavofyork: i've gone with GPLv2.[19]

Gavin Wood (gavofyork), the major author of the CPP Client, decided upon GPLv2. After Gavin's announcement, the developers continued to argue on Github[20] about the appropriate licence for the CPP Client. One developer asserted that the GPLv2 was 'too restrictive'. Vitalik Buterin suggested CC0 Another developer stated that CC0 goes 'too far' and suggested the use of, the Creative Commons Attribution Licence (CCBY).[21] Although Gavin Wood initially adopted GPLv2, he quickly switched to 'GPLv3 and future versions' (GPLv3+). This dialog continued with Gavin Wood promising in January 2014 that the 'license will be changed to something more liberal in due course'. In a posting on 11 July 2014, Nick Savers stated that Vitalik Buterin had selected 'MIT' for the CPP Client on 1 February. However, Gavin Wood responded that 'license is and has always been GPL' on 15 October 2014.Yet, the dialogue continued with a developer on 7 December 2014, stating:

> When is due course?
> But now the issue is closed. If you actively want to hinder the adoption of this code, keep the GPL license. Many companies will not take the risk of building their solutions on GPL code, or likely can't even find a business case for it all. If you want to protect the direction of the code, then at least use LGPL. (Though MIT is clearly the simplest license to understand.)
>
> For us GPL is a no go—we would have to use the Go or Python or Java implementation, even though C++ would fit much better.

[19] <https://github.com/ethereum/aleth/issues/3> accessed 10 December 2020.
[20] <https://github.com/ethereum/aleth/issues/3> accessed 10 December 2020.
[21] <https://creativecommons.org/licenses/by/4.0/> accessed 17 July 2022.

> However, realistically, probably no one is going to implement a clean room implementation from scratch in C++. If strict clean room principles are employed, the team doing it can't even look at your C++ code, unless they want to use GPL too.[22]

Gavin Woods responded the next day to the question of 'when is due course' with '[a]fter the PoC, release series. Specifically alpha or beta. At present, I expect we will move the core to LGPL.' However, this change was never made.

In 2016, Bob Summerwill started a campaign to change the licence from GPLv3+ to a permissive licence. During the discussions, several developers noted that they had signed an agreement to shift the licence of the CPP Client to MIT in August 2015, but the documents appeared to have been lost and a number of the developers had not signed it. Bob started his formal campaign in May 2016 with a message to the developers in the CPP Client:

> As you are probably aware, efforts were made in 2015 to clarify the licensing of various components within Ethereum, namely liberalizing the core to encourage the broadest possible adoption for Ethereum. We never completed that effort.
>
> The licensing for cpp-ethereum itself has flip-flopped a few times and we aren't in a particular clear state right at the moment. To my knowledge we have never had a Contributor License Agreement (<https://en.wikipedia.org/wiki/Contributor_License_Agreement>) as is standard on many FOSS [Free and Open Source Software] projects.
>
> The purpose of a CLA is to ensure that the guardian of a project's outputs has the necessary ownership or grants of rights over all contributions to allow them to distribute under the chosen licence.
>
> With more and more projects looking to build on top of Ethereum, it is important that we have appropriate licensing and most important CLARITY of licensing:-)
>
> In particular, we have an opportunity for Ethereum to become a foundational piece of Hyperledger, following Vitalik's very successful presentation to the Hyperledger Tech Steering Committee in April, but that cannot happen while we have ambiguity of licensing and copyright.[23]

On 26 August 2016, he announced that he was circulating the documents to developers to change the licence to Apache Software Licence version 2. He then spent the next five months trying to get the developers to sign the necessary documents to change the licence. However, he was not successful, in part because Gavin Wood,

[22] <https://github.com/ethereum/aleth/issues/3218> accessed 10 December 2020.

[23] <https://github.com/ethereum/aleth/issues/575> accessed 10 December 2020.

the largest copyright holder in the CPP Client, would not agree to sign the transition documentation.

22.7 Ethereum Client Licence Analysis

We will now turn to analysis of the scans of the four major Ethereum clients: (i) CPP Client; (ii) Go client; (iii) Parity client; and (iv) HyperLedger Besu client.

22.7.1 CPP client

The scan of the CPP Client is for Version 1.8.0 and was performed on September 4, 2020. As noted above, the CPP Client was one of the first clients developed for Ethereum. Gavin Wood started the work and did most of the initial coding. The project licence is GPLv3+. The choice of this licence was subject to extensive discussion and attempts to change it to a permissive licence as discussed above. The project includes 84 per cent third-party Open Source software. The project has 28,298 files of which 23,703 files are third-party Open Source software. These third-party Open Source files have 123 different third-party Open Source components under thirty different Open Source licences. The Black Duck report indicated that 25 per cent of the components are licenced under licences which could potentially cause conflicts. Under the BlackDuck analysis, 350 files (within seven components) have a 'broader reach' licence conflict and four files within one component have a 'narrower reach' licence conflict. The CPP Client also includes 4,401 files within twenty-three components that have potential 'component conflicts'. The project includes some unusual licences such as the Licence for Caramel (a licence from Trustees of the University of Illinois and University of Notre Dame) which requires a notice in the documentation and the 'Crypto Licence' which provides that twelve individuals agree to place twenty projects in the public domain.

22.7.2 Parity client

The scan of the Parity Client is for Version 2.6.8 and was performed on 4 September 2020. After Gavin Wood left the group of founders of Ethereum and his role as Chief Technical Officer of the Ethereum Foundation, he helped start Parity Technologies (Parity) in 2015. Parity developed this new Ethereum client. The project licence is GPLv3. The project includes 88 per cent third-party Open Source software. The project has 11,364 files of which 9,980 files include third-party Open Source software. These third-party Open Source files have 361 different third-party Open Source components under twenty-three different Open

Source licences. BlackDuck noted that 9 per cent of the components are licensed under licences which could potentially cause conflicts. Under the BlackDuck analysis, 158 files (within twelve components) have a 'broader reach' licence conflict and forty-five files within four components have a 'narrower reach' licence conflict. The Parity Client also includes 1,708 files within sixteen components that have potential 'component conflicts'. The licences for the project include many traditional Open Source licences, but it includes one unusual licence, the Unicode Licence for Data Files and Software 2004.

22.7.3 Go client

The scan of the Go Client is for Version 1.9.9 and was performed on 4 September 2020. The Go client (now named 'Geth') was one of the first three Ethereum clients. It was developed at the same time as CPP Client. Jeffrey Wilcke started the work and did most of the initial coding. Jeffrey Wilcke worked with Gavin Wood to ensure that both clients worked on the Ethereum network. The project licence is LGPL version 3 or later versions (LGPLv3+'. The project includes 77 per cent third-party Open Source software. The project has 13,379 files of which 10,263 files include third-party Open Source software. These third-party Open Source files have 106 different third-party Open Source components under twelve different Open Source licences. Black Duck analysis noted that 14 per cent of the components are licenced under licences which could potentially cause conflicts. Under the Black Duck analysis, 100 files (within one component) have a 'broader reach' licence conflict but no 'narrower reach' licence conflict. The Go Client also includes 5,133 files within fourteen components that have potential 'component conflicts.'

22.7.4 Hyperledger Besu client

The scan of the Hyperledger Besu Client (Besu Client) is for Version 1.4.0 and was performed on 23 January 2020. The Besu Client, originally named Pantheon, was developed by the Pegasys team, an affiliate of Consensys. The Pantheon project was developed to make the Ethereum blockchain more attractive to enterprises by using the Java language and adopting the permissive Apache Software Licence. In an interview with Coindesk, Faisal Khan, head of strategy and business development at Pegasys, stated: 'When I used to work on the consulting side of the house at ConsenSys, basically legal departments would just throw up a roadblock if you try to use a GPL in production.'[24]

[24] <https://www.coindesk.com/ethereum-software-client-enterprise-pantheon-pegasys-consensys> accessed 10 December 2020.

After its launch, Pegasys donated Pantheon to the Hyperledger Project of the Linux Foundation, and it was renamed Hyperledger Besu. The project licence is Apache Software Licence version 2. The project includes less than 1 per cent third-party Open Source software. The project has 2,714 files of which five files include third-party Open Source software under six different Open Source licences. These five files have thirty-two different third-party Open Source components. The Open Source licences for the components are virtually all permissive licences except for OpenJDK which is licensed under GPLv2 with the Classpath exception. Under the BlackDuck analysis, one file (within one component) has a 'broader reach' licence conflict. The Hyperledger Besu Client does not have any files with potential 'component conflicts'.

22.7.5 EOS

EOSIO is the blockchain software developed by a private company, Block.one. The EOSIO software is based on a white paper published on 17 September 2017. Block.one hosted one of the largest Initial Coin Offerings (ICOs), selling over 900 million tokens, raising several billion dollars. The Securities and Exchange Commission (SEC) asserted that the ICO was an unregistered securities offering and Block.one settled with the SEC for $24 million in 2019. Version 1 of the EOSIO software was released on 2 June 2018.

The scan of the EOSIO software is for Version 2.0.0 and was performed on 4 September 2020. The EOSIO software is licensed under the MIT licence. The project includes 42 per cent third-party Open Source software. The project has 13,799 files of which 5,735 files include third-party Open Source software. These third-party Open Source files have ninety-nine different third-party Open Source components under twenty-four different Open Source licences. Black Duck noted that 23 per cent of the components are licensed under licences which could potentially cause conflicts. In addition, two components were from a proprietary third party. Under the BlackDuck analysis, none of the projects have 'broader reach' licence conflict or 'narrower reach' licence conflict. The EOSIO software also includes 2,013 files within twenty-three components that have potential component conflicts. Although the EOSIO project licence is the permissive MIT licence, the project includes a number of components under copyleft licences including GPLv2 and GPLv3.

22.7.6 Corda

Corda is the blockchain software developed by a private company, R3, and, initially, a consortium of financial institutions. Although the Corda software was initially released in April 2016, it was quickly released as an Open Source licence on 30 November 2016. At the time, David Rutter, the CEO of R3, noted:

> Blindly investing millions of dollars in small, disparate technology projects is not appropriate for banks at a time when budgets are stretched. The risk of backing the wrong horse could far outweigh the potential gains. Given that the power of this technology lies in its network effect, the consortium model is the ideal method to get it off the drawing board and into the wholesale financial markets.[25]

The scan of the Corda software is for Version 4.4 and was performed on 23 January 2020. The Corda software is licensed under the Apache Licence version 2 and includes very little third-party Open Source software. The project includes 5 per cent third-party Open Source software. The project has 3,235 files of which 159 files include third-party Open Source software. These third-party Open Source files come from 168 different third-party Open Source components under twenty-three different Open Source licences. Synopsys noted that 9 per cent of the components are licensed under licences which could potentially cause conflicts. Under the BlackDuck analysis, four files (within three components) have a 'broader reach' licence conflict and seven files (within seven components) have a 'narrower reach' licence conflict. The Corda software also has four files within six components that have potential 'component conflicts'. Although the Corda project licence is the Apache Software Licence version 2, the project includes some files under GPLv2 with the Classpath exception, LGPL version 2.1 and LGPL version 3.0.

22.8 Conclusions

Open Source software is critical to the success of the blockchain ecosystem, but it has not received much attention. This review of project licence selection and bill of materials results in two major conclusions.

The first conclusion is the difference in approach relating to project licence selection for the Bitcoin and Ethereum clients as well as the blockchain software developed by corporations, EOS and Corda. The project licences for the initial Ethereum clients, the CPP Client, Parity Client, and Go Clients were copyleft licences, the most restrictive licences. Although copyleft licences are widely used outside of the blockchain markets, they impose significant compliance problems because of their uncertain scope and, consequently, are viewed as risky by corporations. The discussion of project licence selection for the CPP Client which is

[25] <https://www.reuters.com/article/idUSKCN12K17E> accessed 10 December 2020.

recorded in Github is remarkable for the lack of reference to the experience of other Open Source projects in the last thirty years as well as the lack of any advice from lawyers. The selection of GPLv3 + for the CPP Client was very unusual because the GPLv3 licence has not been widely adopted due to its complexity. In my experience, the discussion echoes licence selection discussions for projects in the early 2000s. The last twenty years has resulted in most Open Source software projects which need corporate adoption for success. Particularly, 'infrastructure' or platform software is licensed under a permissive licence with the Apache licence being very popular. The reluctance of corporations to adopt Ethereum due to licence considerations became sufficiently serious that Consensys, the Ethereum-focused conglomerate, arranged to have a new Ethereum client developed so it could be licensed under a permissive licence. This client, the Hyperledger Besu client (originally the Pantheon client), was licensed under the Apache licence. Faisal Khan, head of strategy and business development at Pegasys, was very direct about this goal in an interview with Coindesk: 'When I used to work on the consulting side of the house at ConsenSys, basically legal departments would just throw up a roadblock if you try to use a GPL in production.'[26] The project licence for Bitcoin Core is the permissive licence, MIT. MIT appears to be the preferred licence of many Bitcoin clients and their forks. We do not have a record of how this selection was made, but it significantly reduces the complexity of licence compliance. The project licence selection for corporate developed platforms, such as EOS and Corda, reflect an understanding of the challenges of copyleft licences. Both platforms are licensed under 'permissive' licences, MIT for EOS and Apache for Corda.

The second and more challenging issue is licence compliance in a 'decentralised' platform. This issue is particularly serious for the three initial Ethereum clients which are licensed under copyleft licences which require more careful attention to licence compliance than permissive licences. Once again, the results are different between the 'decentralised' software platforms such as the Bitcoin clients and the first three Ethereum clients and the corporate developed software. The decentralised software platforms, such as the Bitcoin clients, have a significant amount of third-party Open Source projects: 78 per cent and 83 per cent. The Bitcoin clients also have a significant number of potential conflicts: Bitcoin SV (19 per cent) and Bitcoin Core (16 per cent). Similarly, the three earliest Ethereum clients also included very significant amounts of third-party Open Source projects with 77 per cent, 84 per cent, and 88 per cent. They also have a significant number of potential licence conflicts with the CPP Client (25 per cent), Parity Client (9 per cent), and Go Client (14 per cent). On the other hand, the corporate developed Ethereum client (developed by Pegasys) and the one of two corporate developed blockchain platforms had a much lower amount of third-party Open Source projects: Hyperledger

[26] <https://www.coindesk.com/ethereum-software-client-enterprise-pantheon-pegasys-consensys> accessed 10 December 2020.

Besu (1 per cent) and Corda (5 per cent). They also have lower percentages of potential licence conflicts: Hyperledger Besu (only one potential conflict) and Corda (9 per cent). EOS is an outlier in this group of blockchain software developed by corporations with 42 per cent third-party Open Source projects and 24 per cent potential licence conflicts.

This chapter has not addressed the potential issues arising from the interaction of the software at Level 1 with the software implementing Level 2 functionality and DApps. The blockchain ecosystem is complex and multilayered with a strong dependence on Open Source software. This complexity and the decentralised nature of the blockchain projects means that Open Source licence selection and compliance are likely to be important issues in the blockchain ecosystem for the foreseeable future.

23
Open Hardware

Andrew Katz

23.1 Introduction

Open hardware has taken a slower and more cautious route to success than Open Source software, and in many ways lags behind it. At this juncture, there are far fewer businesses operating in the world of open hardware, and those businesses are less prominent and successful (although there are signs that this is rapidly changing). There are also far fewer specific open hardware licences than Open Source licences (which many will say is a good thing), and there are still fundamental questions about the applicability of various different forms of intellectual property right (IP) to different aspects of hardware.

23.2 What is Hardware?

Hardware can cover everything from printed circuit boards to silicon chip designs, to cases for computer hardware, to mechanical devices, artistic objects, and even liquids like cola or beer. Ultimately, some physical matter—atoms—must be involved. Whereas information, software, and content as intangible assets can remain

Andrew Katz, *Open Hardware* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press.
© Andrew Katz 2022. DOI: 10.1093/oso/9780198862345.003.0023

in the abstract domain[1] (in some cases, until the instant they are used or consumed by someone), it is a characteristic of hardware that it consists of physical material.

23.3 A Brief History

The concept of 'openness' only became necessary to counterbalance societal or legislative processes which began to apply restrictions such as copyright or patent—to 'close'—what had inevitably, in its embryonic stages, been open. As explored in Chapter 24, 'openness' (in the sense of freedom of use) only becomes an issue for hardware, as with any asset class, once the law or norms restricted the use of that hardware through the imposition of IP. As we will see shortly, IP affects hardware, although the scope and effect of coverage can be quite different from the way in which IP affects software.

The evident success of Open Source instigated the exploration of a similar concept for hardware: *open hardware* or *open source hardware*.[2] For example, Bruce Perens, one of the founders of the Open Source Initiative (OSI), established the Open Hardware Certification Program in 1997.[3] This was a process which allowed organisations providing electronic devices to self-certify that the interfaces of their designs were documented in order to facilitate the programing of device driver software. The focus, therefore, was not so much on the design documentation for the device itself but the ability for the device (e.g. a printer) to be accessible to a broad range of software. This would give comfort to purchasers of the equipment that it would (at least in theory) be possible for them to continue to use their equipment should they change their operating system, or if the manufacturer of the equipment went out of business and no longer provided software updates.[4]

Other initiatives focused more on the design of the hardware itself, such as FreeIO and the OpenGraphics project, and, in parallel, the growing *maker*

[1] More recently, this is usually digital, but information can exist in analogue form which is still abstract such as a series of fluctuations in the magnetic domains on a tape medium such as a compact cassette.

[2] Although the definition of 'open source hardware' promulgated by the Open Source Hardware Association (see later in the chapter), there is no such consistent definition for 'open hardware'. Originally, 'open hardware' was used for hardware with freely available interface information (even if there was no access to the designs for the hardware coupled with a right to modify and recreate them) and 'open source hardware' for hardware where the designs were available for copying, modification, making, and distributing the product, and redistribution of the product and the design. The definitions are now somewhat blurred: for example, the CERN Open Hardware Licence is very much intended to allow the designs to be made available, so to that extent it may also be considered to be an Open Source Hardware licence.

[3] <https://web.archive.org/web/19981212031618/http://www.openhardware.org/> accessed 20 April 2022.

[4] See n 3.

movement, meetups, and groups developed such as the Open Source Hardware Camp[5] and OGGCamp, both in the UK. In 2010, the first Open Hardware Summit was held in New York, and the germ of an open source hardware definition was established, and developed throughout that year, with version 1.0 finally published in early 2011.[6] The Open Source Hardware Association has proven to be longer lasting than many previous open hardware associations, and held its 10th and 11th summits (albeit virtually, owing to COVID-19) in 2020 and 2021. As well as promoting the Open Source Hardware Definition, and holding an annual summit, it has a certification program which allows organisations providing Open Source Hardware (OSHW) to obtain a certification that their designs are compliant.[7] At the time of writing, 1,484 hardware designs were certified.[8]

Between 2010 and 2022, open hardware has matured, with the growing success of projects such as Arduino (a family of open source hardware microcontroller boards), RepRap (a 3D printer which is itself available as open hardware), and, more recently, open source chip designs such as those based on the RISC-V instruction set architecture. The BBC launched a low-cost open hardware microcontroller aimed at education (the micro:bit) in 2015 with the first units delivered to children in 2016.

Open hardware also gained ground in research and academia: as well as RepRap mentioned earlier, which was developed by Adrian Bowyer at Bath University, CERN launched the White Rabbit Project, a mechanism for timing events to sub-nanosecond accuracy using the well-understood Ethernet networking protocol as its base. The project was initiated as a result of proposals presented by Javier Serrano in 2008, and also led to launch of the CERN Open Hardware Repository and the launch of the first version of the CERN Open Hardware Licence in 2011.[9]

A particularly interesting area of open source hardware is open source silicon. Silicon chip designs have been released under Open Source licences for some time (e.g. Sun Microsystems released OpenSPARC under the GNU General Public License version 2.0 (GPLv2) in 2005). Since then, several chip designs have been released under Open Source licences,[10] and companies such as Western Digital Corporation and SiFive have developed products using microprocessor core designs based on the Risc-V instruction set architecture. SiFive has at the date of writing received a total of US$365.5 million in investment, giving it an overall

[5] Now incorporated into the splendidly named Wuthering Bytes tech festival which takes place in Yorkshire in the UK every year (with a name like that, where else?). <https://wutheringbytes.com/> accessed 20 April 2022.

[6] <https://freedomdefined.org/OSHW> accessed 20 April 2022.

[7] <https://certification.oshwa.org> accessed 20 April 2022.

[8] <https://certification.oshwa.org/list.html> accessed 20 April 2022.

[9] Javier Serrano is still involved in both projects and is a member of the core drafting team for the CERN OHL.

[10] A Katz 'A Survey of Open Processor Core Licensing', JOLTS Vol 10.1 <https://www.jolts.world/index.php/jolts/article/view/130> accessed 20 April 2022.

market value of over US$1 billion at the share value for the most recent funding round.[11]

Perhaps the most significant indicator that open source hardware has come of age is that the European Commission in 2019 issued a call for tenders for a research project with a scope which explicitly covers open hardware, to examine technological independence, competitiveness, and innovation in the EU economy.[12] The final report was delivered in March 2021.

23.4 The Open Source Hardware Definition

The Open Source Hardware Definition and its associated Statement of Principles rely heavily on the Open Source Definition (OSD) developed by the OSI, and the Four Freedoms of the Free Software Foundation, respectively. It is a nice illustration of the close relationship between these two venerable Open Source definitions that they can be combined into, effectively, a single unified definition for open source hardware (OSHW).[13]

23.4.1 The Open Source Hardware Statement of Principles

> Open source hardware is hardware whose design is made publicly available so that anyone can study, modify, distribute, make, and sell the design or hardware based on that design. The hardware's source, the design from which it is made, is available in the preferred format for making modifications to it. Ideally, open source hardware uses readily-available components and materials, standard processes, open infrastructure, unrestricted content, and open-source design tools to maximize the ability of individuals to make and use hardware. Open source hardware gives people the freedom to control their technology while sharing knowledge and encouraging commerce through the open exchange of designs.

23.4.2 The Open Source Hardware Definition

> Introduction
>
> Open Source Hardware (OSHW) is a term for tangible artifacts—machines, devices, or other physical things—whose design has been released to the public in such a way that anyone can make, modify, distribute, and use those things. This

[11] <https://www.crunchbase.com/organization/sifive> accessed 17 June 2022.

[12] <https://ec.europa.eu/digital-single-market/en/news/call-tenders-study-impact-open-source-software-and-hardware-technological-independence> accessed 20 April 2022.

[13] <https://www.oshwa.org/definition/> accessed 20 April 2022.

> definition is intended to help provide guidelines for the development and evaluation of licenses for Open Source Hardware.

Hardware is different from software in that physical resources must always be committed for the creation of physical goods. Accordingly, persons or companies producing items ('products') under an OSHW license have an obligation to make it clear that such products are not manufactured, sold, warrantied, or otherwise sanctioned by the original designer and also not to make use of any trademarks owned by the original designer.

The distribution terms of Open Source Hardware must comply with the following criteria:

1. Documentation

The hardware must be released with documentation including design files, and must allow modification and distribution of the design files. Where documentation is not furnished with the physical product, there must be a well-publicised means of obtaining this documentation for no more than a reasonable reproduction cost, preferably downloading via the Internet without charge. The documentation must include design files in the preferred format for making changes, for example the native file format of a CAD program. Deliberately obfuscated design files are not allowed. Intermediate forms analogous to compiled computer code—such as printer-ready copper artwork from a CAD program—are not allowed as substitutes. The license may require that the design files are provided in fully-documented, open format(s).

2. Scope

The documentation for the hardware must clearly specify what portion of the design, if not all, is being released under the license.

3. Necessary Software

If the licensed design requires software, embedded or otherwise, to operate properly and fulfill its essential functions, then the license may require that one of the following conditions are met:

a) The interfaces are sufficiently documented such that it could reasonably be considered straightforward to write open source software that allows the device to operate properly and fulfil its essential functions. For example, this may include the use of detailed signal timing diagrams or pseudocode to clearly illustrate the interface in operation.

b) The necessary software is released under an OSI-approved open source license.

4. Derived Works

The license shall allow modifications and derived works, and shall allow them to be distributed under the same terms as the license of the original work. The license shall allow for the manufacture, sale, distribution, and use of products created from the design files, the design files themselves, and derivatives thereof.

5. **Free redistribution**

The license shall not restrict any party from selling or giving away the project documentation. The license shall not require a royalty or other fee for such sale. The license shall not require any royalty or fee related to the sale of derived works.

6. **Attribution**

The license may require derived documents, and copyright notices associated with devices, to provide attribution to the licensors when distributing design files, manufactured products, and/or derivatives thereof. The license may require that this information be accessible to the end-user using the device normally, but shall not specify a specific format of display. The license may require derived works to carry a different name or version number from the original design.

7. **No Discrimination Against Persons or Groups**

The license must not discriminate against any person or group of persons.

8. **No Discrimination Against Fields of Endeavor**

The license must not restrict anyone from making use of the work (including manufactured hardware) in a specific field of endeavour. For example, it must not restrict the hardware from being used in a business, or from being used in nuclear research.

9. **Distribution of License**

The rights granted by the license must apply to all to whom the work is redistributed without the need for execution of an additional license by those parties.

10. **License Must Not Be Specific to a Product**

The rights granted by the license must not depend on the licensed work being part of a particular product. If a portion is extracted from a work and used or distributed within the terms of the license, all parties to whom that work is redistributed should have the same rights as those that are granted for the original work.

11. **License Must Not Restrict Other Hardware or Software**

The license must not place restrictions on other items that are aggregated with the licensed work but not derivative of it. For example, the license must not insist that all other hardware sold with the licensed item be open source, nor that only open source software be used external to the device.

12. **License Must Be Technology-Neutral**

No provision of the license may be predicated on any individual technology, specific part or component, material, or style of interface or use thereof.

These definitions, like the Four Freedoms and the OSD on which they are based, focus very much on licensing and, accordingly, they rely heavily on the underlying IP applicable to hardware.[14] However, hardware provides some significant

[14] These principles should be interpreted consistently with the equivalent principles in the Open Source Definition: see C hapters 1, 3, and 16 for more details.

challenges, as the intellectual property regime which potentially applies to it is somewhat more complex than that applicable to software.

23.5 Hardware and Reciprocity (Copyleft)—Intellectual Property

Primarily, software is governed by copyright (it is generally treated by copyright law in a similar way to a literary work). Patents and other IP such as database right can also affect software, but IP applies to hardware in a more complex and inconsistent way. An upshot of this is that it is more difficult to apply reciprocal licensing obligations to hardware designs. Note that it is becoming common to use the term 'reciprocal' in preference to 'copyleft' in relation to open hardware licensing because hardware is dependent on a broader range of intellectual property rights than copyright alone, so it is potentially misleading to use the term 'copyleft' with its close association to copyright. In this chapter, you can regard the terms as broadly interchangeable.

As explained in Chapter 3, reciprocity relies on there being some form of IP which can apply each time an item is distributed. For copyleft, the relevant form is copyright. Every time IP impinges there is the opportunity to apply a condition to the licence which is being granted, and reciprocity works because the condition is that the item being distributed must itself be licensed on the same reciprocal terms. This perpetuates the licence each time the relevant item (or a derivative of it) is distributed.

If the act of distribution (or any of the necessary precursors to distribution, such as copying or making the item available to the public) requires no licence under any IP, then the distribution of the item cannot be controlled through application of a conditional licence to IP because there is no opportunity to apply a condition (such as the one implementing copyleft).

To take an extreme example: it is obvious that a work, such as the *Great Gatsby* (which is now in the public domain because the copyright in it has expired) cannot be subject to any copyright licence, so it is not possible to apply any copyleft restrictions to it.[15]

[15] Richard Stallman (founder of the Free Software Foundation) was aware of this when he criticised the Swedish Pirate Party's recommendations for a very short period of copyright (five years). At the end of five years, under its plans, computer software would enter the public domain and therefore what was previously available under the GPL would be capable of being incorporated into proprietary software without the requirement for the corresponding source being made available. In other words, the core purpose of the GPL—to ensure that free software remained free—would be defeated after five years. This offended Stallman's anti-closure view that free software must remain free, and after considering (and rejecting) a special exception for software which would *extend* the Pirate Party's proposed five-year term of copyright for the specific purpose of allowing the GPL's copyleft mechanism to continue working for longer, he concluded that it would be better for the source code for non-free software to be placed in escrow and automatically released at the end of the five-year term.

When a piece of software is distributed, it will almost invariably have been copied and then made available to the public before the distribution takes place. Any one of these acts is likely to be and can be presumed to be affected by copyright, and therefore performing that act will require a licence and consequently provide an opportunity to apply a copyleft condition.

With hardware, it is not necessarily the case that making an item to a design, or making the item available to the public, or physically transferring the item to someone else, requires some form of licence under which the condition can be applied. An example of this is the OpenCola. As its name suggests, OpenCola is a Coca-Cola-style drink, differing significantly from Coke in that its recipe is publicly available[16] and licensed under the GNU General Public License (GPL). However, because making a drink to a recipe is not an act which is restricted by copyright, doing so does not require a licence and therefore provides no opportunity for the licensor of the recipe to require anyone making OpenCola (or a variant) to make the recipe available to them. If OpenCola was software source code, then the act of compiling the source to make the executable software would be creating an adaptation (or depending on the jurisdiction's terminology, a derivative work) which would be an act restricted by copyright, for which a licence (in this case, the GPL) is required. In this example, the GPL's copyleft condition would therefore kick in, and the licensee would be required to provide (or offer to provide) a copy of the source code. For OpenCola, the copyleft effect would not extend to the product itself.[17]

In brief, whereas almost any activity involving software—running it, distributing it, copying it, amending it—will involve an act reserved to the copyright owner, it is by no means clear that similar acts relating to hardware are similarly controlled.[18]

Some major open hardware licences, the CERN Open Hardware Licence family[19] (other than in the case of CERN, the permissive variant), and the TAPR Open Hardware License,[20] seek to apply a form of reciprocity. The lack of impingement of IP on hardware causes issues for both licence models, as will be discussed shortly.[21] They use very different techniques to achieve their aim.

[16] <https://web.archive.org/web/20010218075323/http://www.opencola.com/download/3_softdrink/formula.shtml> accessed 20 April 2022.

[17] It would extend to the text of the recipe though: if the licensee transfers the recipe text so someone else, that recipient would have the right to change it, and distribute it, and if they did so, that distribution would also have to be subject to the GPL.

[18] Except in relation to patent, which is dealt with later in this chapter.

[19] <http://www.ohwr.org/projects/cernohl/wiki> accessed 20 April 2022.

[20] <http://www.tapr.org/ohl.html> accessed 20 April 2022.

[21] For further analysis of these issues, see A Katz, <http://www.ifosslr.org/ifosslr/article/view/69 (2012), and Richard Stallman (1999) <http://www.linuxtoday.com/infrastructure/1999062200505NWLF> both accessed 20 April 2022.

Application of reciprocity to open source hardware is not only problematic from an IP perspective. There are also challenges from an economic and practical perspective.

23.5.1 Reciprocity and the costs of reverse engineering

The cost of making a bit-for-bit copy of a piece of software is close to zero. In contrast, because hardware involves physical material, there will always be some cost involved in instantiating a piece of hardware. For example, even if the hardware can be replicated by 3D printing, the cost of the feedstock needs to be considered. Many hardware designs will be too complex to be replicated using a 3D printer: for example, the front suspension sub-assembly for a car may require milling and machining of the steel components which make it up. Even if the design documentation includes CNC (computer numerical control) files to control the machine tools such as a lathe and milling machine, the effort required in setting the machines up, monitoring them, and finishing and assembling the mechanical components is significantly greater than the effort required to compile the binary of a piece of software from the source code or, even more simply, copying an existing binary.[22]

If a piece of software is covered by a copyleft licence, such as the GPL, someone wishing to make use of the functionality of that software has two options from a copyright perspective.[23] They can either agree to comply with the terms of the GPL, and simply copy the software, essentially for zero cost, accepting the copyleft (and other) requirements of the GPL, or they can decide to expend engineering effort to replicate the functionality of the software by reverse engineering and recreating it.

Recall that copyright protects the expression of an idea and not the underlying idea itself (although that may potentially be subject to patent protection).[24] Thus it is possible to determine the functionality (idea) of a piece of software, and then replicate that software independently (using a different expression), so that the rights in the new piece of software belong to its author, and it may be exploited freely by that author without reference to the rights holders of the original piece of software[25] (assuming no patent issues). Compaq famously employed a reverse-engineering

[22] Other issues that may be significant are the capital cost of the equipment itself, physical space required to house it, the environment (e.g. humidity, temperature, security), energy supply, and so on. We explore these in brief later in this chapter.

[23] For completeness, two further options are to ignore the licence and infringe the copyright (with the legal consequences that that might entail), or to persuade the copyright owners of the software to grant a licence which is more amenable (possibly at a price)—something Richard Stallman calls buying exceptions to the GPL.

[24] This is somewhat of an over simplification, since it ignores rights that may exist in the structure, sequence, and organisation of the code and its equivalent in other jurisdictions.

[25] Ignoring, for the time being, other rights, particularly patent rights, that may cover the original software.

and rewrite (a 'clean room') technique to replicate the BIOS[26] of the original IBM PC. This enabled Compaq to create the first legal IBM PC clones.

A similar reverse-engineering technique can be employed with hardware, such as mechanical and electronic devices.[27]

To take an extreme example, there is a monumental difference in cost between taking a copy of the Linux kernel (almost zero) and in replicating its functionality by reverse engineering and recoding it. Even in 2008, when the Linux Kernel was vastly simpler than it is now, the cost of coding the kernel from scratch was estimated by the Linux Foundation at $1.4 billion.[28] It is not surprising that businesses opt to comply with the GPL rather than try to recreate their own compatible kernel.[29]

The economics for hardware are liable to be different. Given that the instantiation of any piece of hardware is liable to require non-trivial effort in any event, the differential in cost between re-engineering a design in a way which does not impinge on any intellectual property rights, and using an existing open hardware design (and agreeing to comply with the conditions applicable to it) is likely to be smaller than the equivalent scenario in software. The extent to which this is true will vary significantly between types of hardware: it is less likely to be true for devices such as microprocessor designs than it is for mechanical suspension parts, for example.

Therefore, before seeking to apply a copyleft (or other form of reciprocal) licence to hardware, it is important to consider the characteristics of the hardware being licensed: the additional complexity imposed by a copyleft licence may not be justified if it is trivially easy to design around the requirements in any event.

23.5.2 The boundary problem

If copyleft is to work effectively in open hardware, there must be some clear limit to the degree to which the copyleft element of a design is intended to affect the rest of

[26] Basic Input Output System. This is a relatively small piece of software embedded in the read-only memory of a PC which provides a basic hardware interface to the computer's memory and peripherals, and an interface to the operating system, such as Windows or (later) Linux. If a PC from a manufacturer other than IBM could be made to run a BIOS which was functionally compatible with the IBM PC BIOS, this would enable it to run software, including the operating system, which was originally written for the IBM PC.

[27] As we have seen, it is by no means clear that the hardware itself is covered by copyright. Accordingly, the process of creating an alternate design containing the same functionality, free of any IP, is relatively more straightforward for hardware than for software. Furthermore, it is even simpler when open hardware is concerned, because the design documentation will be available.

[28] <https://www.linuxfoundation.org/press-release/2008/10/linux-foundation-publishes-study-estimating-the-value-of-linux/> accessed 20 April 2022.

[29] Another option is for businesses needing a Unix-like kernel who are not keen on using software covered by the GPL to adopt FreeBSD, which is licensed under the more liberal BSD licence.

the design. For example, if a wheel hub is released under an open source hardware licence, does that mean that if it is included in a front suspension sub-assembly, that whole of that sub-assembly will become subject to the copyleft licence? What if that sub-assembly is incorporated into a car: does that copyleft impact the whole car and mean that the design for the whole car must be released under the same licence? These issues exist for Open Source as well, but the boundaries are reasonably well understood (even so, they continue to generate a great deal of debate). The definitions are somewhat easier in the software world—the Mozilla licence, for example, is intended to apply file-level copyleft, where the term 'file' is reasonably well understood; at least it is much better understood than a vague term like 'sub-assembly'.

The boundary problem also applies in terms of the components of which a design is composed. What level of detail is required? To take the example of a wheel hub once more: the hub is likely to require components such as a ball bearing assembly. Ball bearings are available in a number of common sizes and specifications and consist of two concentric hardened steel rings (races) with a number of hardened steel balls in between them. A reciprocal open source hardware licence may require that the complete design materials for the product are required. Strict interpretation could require that the wheel hub design also contains complete documentation for manufacturing the ball bearing, which in turn would require complete documentation for manufacturing the steel balls. To take this to a ridiculous extreme, instructions would be required for the manufacture of the entire hub assembly from its constituent atoms.[30] This is not a problem (generally) with software because, ultimately, the source code provides enough information to compile the object code of the software: this is equivalent to saying that there is enough information for the software to be manufactured out of the 'atoms' which software is made of—the binary digits 1 and 0.

By considering both the tangible nature of hardware and the breadth of the range of classes and categories of item to which the term 'hardware' can apply, unlike the much simpler category of software covered in Open Source licensing, the additional complexities become clear.

The reciprocal variants of the current version of the CERN Open Hardware Licence (version 2) address this issue by exempting from the requirement to provide the complete design documentation, any components which qualify as 'Available Components'. We look at this mechanism in greater depth below.

[30] An explanation for this oversight may be that many people think of open hardware as being mainly electronic devices. Electronic construction tends to consist of components, such as resistors, capacitors, ICs, transistors, and inductors, all being soldered onto a circuit board. The components are all standard items, with well understood specifications, and therefore, it is fairly clear that the level of abstraction required of the design is at component level.

23.5.3 The competing copylefts problem

It is clear that if a copyleft licence requires derivatives to be released under exactly the same licence, then it becomes impossible to combine two works into a third, being a derivative of both the original works, if each of the original works is subject to a different copyleft licence. For example, it is not possible to license a work consisting of combined GPL and OSL (Open Software License) components under *only* the GPL (as required by the GPL) and *only* the OSL (as required by the OSL). The requirements of the two licences cannot be satisfied simultaneously.

This is a well-known problem for Open Source, with compatibility lacking between even different versions of the GPL.[31]

The practical issue is that the effectiveness of an open project is understood to grow, as a network effect, in proportion to the square of the participants.[32] The set of all software released under a particular copyleft licence can be considered to be a commons, in which all components are freely able to interact. If two commons of software projects under different licences are unable to interoperate, then each commons on its own would be a quarter as effective, in terms of potential interactions between the two commons combined.

Attempts have been made to deal with restrictions on interaction: some Open Source licences explicitly allow relicensing under similar licences. For example, the European Union Public Licence (EUPL) allows relicensing under the GPL. However, care has to be taken in allowing relicensing. If someone has chosen a licence with relatively strong copyleft then they are unlikely to be happy with the ability of anyone downstream to choose a licence which has weaker copyleft, or no copyleft at all.

Therefore, learning from what has gone before in the world of software, there should ideally be only one copyleft Open Source hardware licence (or a family of intercompatible licences), to prevent licence incompatibility through licence proliferation.

23.6 Hardware and Other Forms of Intellectual Property Right

23.6.1 Patents

Copyright has been an effective way of controlling the use and exploitation of software (and the implementation of copyleft), in part because, under the Berne

[31] Unless there is an option to take a later version, for example, GPLv2 or later is compatible with GPL v3.

[32] Metcalfe's law.

Convention, it arises automatically, without the formality of any assertion or registration. Patents, in contrast, may provide a number of opportunities to impinge on the use and exploitation of a piece of hardware but require extensive formality and budget to obtain and maintain. At first glance, it may seem that if patents give a number of opportunities for a licence to impinge during the lifecycle of a piece of hardware that copyright does not, then for a hardware licence to be effective, and, in particular, for it to be able to implement copyleft, it should concentrate on patents rather than copyright.

This assumption has a number of difficulties:

1. It has been noted that a characteristic of the Open Source development model is that there is a low barrier to entry for participants. Each participant in a software project will automatically obtain copyright in the work that he or she submits, and this can form the basis of the licensing applicable to the project, both in relation to third parties and also in relation to the relationship between the participants themselves (as in the case of the Linux kernel, for example), or the participant and the project sponsor. In contrast, patent protection is not automatic, so there has to be a more complex mechanism in place to mediate the relationship between the participants, the sponsor (if any) and the end users. This will have to take into account the issues set out shortly.
2. Even relatively small pieces of software code may attract copyright protection. A smaller number of designs, whether hardware or software, will potentially have the necessary quality of inventiveness and novelty to qualify as a patentable invention.
3. Patents are expensive and take time to apply for. This militates against many open projects, which have minimal funding. It also requires that some mechanism is in place to decide how funding is obtained, and is spent, and which inventions are worthy of being applied for.
4. Patentable inventions need to be kept secret at the initial stages. This means in practice that information about the invention needs to be kept to a small number of people until (dependent upon jurisdiction) it has been filed. This adds complexity (non-disclosure agreements need to be entered into, and the various recipients of the information need to be trustworthy), and, crucially, it is contrary to commonly employed Open Source community development models, as it is directly in opposition to the principles of 'given enough eyeballs, all bugs are shallow' and 'release early, release often'.
5. A patent is only effective in the jurisdiction in which it is granted. Worldwide coverage requires multiple patent applications and grants, and this multiplication rapidly becomes very expensive.

That is not to say that is impossible to establish a reciprocal open development model based on patent protection, but these barriers suggest that it is likely to be more challenging than the equivalent copyright-based Open Source software model.

For example, the secrecy problem may be addressed by having an inner circle of developers who have signed mutual non-disclosure agreements (NDAs), and anyone who comes up with an invention which they feel may be patentable may apply to join the inner circle, and therefore share his or her invention subject to the mutual NDAs.

23.6.2 Design rights, database rights, and other intellectual property rights

Hardware and hardware designs may also be covered by other IP such as design rights (registered and unregistered), semiconductor topography (mask) rights, and database rights (for bill of materials, for example). Many Open Source software licences refer specifically to the IP which are most applicable to software (copyright, and occasionally patent), without mentioning or allowing for these other rights. For this reason, licences drafted to cover hardware, such as the CERN-OHL family and the Solderpad licence, either allow for these additional rights explicitly, or they are carefully drafted so as not to limit themselves to any specific underlying rights.

In common with Open Source software licences, trademarks, if they are mentioned at all, are not freely licensed alongside other IP in open hardware licences. There may be clauses restricting the use of either the licensor's or the licence sponsor's trademarks to suggest that a modification of a particular design or an article made to it has been approved by the licensor or the licence sponsor. Trademarks may be addressed (as sometimes happens in the world of Open Source) through an orthogonal trademark licence which allows the project trademark and variants of it to be used only where certain criteria have been met: for example, where an item meets a particular compatibility standard. The Arduino trademark policy is a good example of this.[33]

23.7 Specific Open Hardware Licences

Although licences specifically intended for open hardware exist, many open source hardware projects are licensed under Open Source software licences (e.g. GPL,

[33] <https://www.arduino.cc/en/trademark> accessed 20 April 2022.

LGPL, BSD, Apache). They are also sometimes licensed under content licences such as one of the Creative Commons family. However, software and content licences are typically not suited to open hardware, for reasons as diverse as inappropriate terminology (e.g. what is the 'object code' of an open hardware design?), the inapplicability of certain types of IP and width of IP that may be applicable as has been considered earlier.

Accordingly, a number of different licences have been developed for use in open hardware. The most prominent open hardware licences are the CERN Open Hardware Licence family,[34] the TAPR Open Hardware License,[35] and the Solderpad Hardware License. Other licences such as the Open Compute Project Hardware Licenses have many characteristics of being an open hardware licence, but the extent to which they qualify as true open hardware licences is up for debate (e.g. the patent licences granted under the Open Compute Project Hardware Licences are restricted to implementations of the design as specified by the original licensor, so from this perspective they are better regarded as standards licences than open hardware licences).

The TAPR licence was developed by attorney and radio amateur John Ackermann, who acknowledges many of the difficulties of applying copyleft to hardware.[36] The TAPR licence is intended to be a copyleft licence, and although it acknowledges copyright in the design documentation, it attempts to cover the impingement issue by acting as a contract and thus contractually binding anyone who relies on the licence to release modifications to the design under the same licence.[37] The contract is formed by presenting an offer which is capable of acceptance by anyone wishing to make use of the licensed invention, and as such presents itself as a unilateral contract, capable of acceptance by conduct, without communication of that acceptance to the licensor.[38] Ackermann is aware of the potential problems in common law jurisdictions which require consideration of contract, for example a payment, especially where the licensor does not have any rights (such as a patent) to license in return. He attempts to deal with this not by granting a licence, which

[34] <http://www.ohwr.org/projects/cernohl/wiki> accessed 20 April 2022.

[35] <http://www.tapr.org/ohl.html> accessed 20 April 2022.

[36] <http://www.tapr.org/Ackermann_Open_Source_Hardware_Article_2009.pdf> accessed 20 April 2022.

[37] A number of mechanisms based on contract have been suggested to make reciprocity work for hardware. The principle is generally that in order to use design A, the licensee enters into a contract with the licensor, and then whenever the licensee distributes an article made to design A (or a derivative), this must be subject to the same licence, which is itself a contract, binding on the next downstream licensee, and so on. It has been suggested that this can be used to create, contractually, pseudo-IP that do not otherwise exist at law. However, a contract is only effective between the parties to it. If someone receives the design, or the documentation, for whatever reason, not subject to the contract, then they will not be a party to the contract, and will not be bound by the pseudo-IP. If the licence is reliant on *real* IP, then someone seeking to exploit the design will have to have some form of licence, and will be in breach unless they do so, irrespective of whether there is a contract with the rights holder. There is therefore no necessity for a contractual chain.

[38] *Carlill* v *Carbolic Smoke Ball Company* [1892] EWCA Civ 1.

he believes would be failed consideration if there is no underlying IP to license, but by granting a patent non-assert, which he claims would be effective as it anticipates, for example, that the licensor might acquire a relevant patent in the future.[39]

The CERN Open Hardware Licences take a slightly different tack. They concentrate on the design documentation, and the user's rights (subject to the conditions applying to the licence) are granted once the user performs any act that would otherwise impinge on an IP in the design documentation. Version 1.2 of the licence, provides that amendment of the design documentation is conditional on the publication of the amendments. However, that obligation is suspended until such time as an instantiation of the design is made available to the public.[40] The 1.x versions of the CERN OHL are reciprocal (copyleft) licences. Version 2 of the CERN OHL is somewhat different.

It comes in three variants, a permissive variant (which we discuss in the section to follow), and two reciprocal (or copyleft) variants, in 'strong' and 'weak' forms (intended to be broadly similar in effect to the GPL and LGPL respectively). As with version 1.2 of the CERN OHL, the primary focus is on licensing the underlying design documentation. As soon as a licensee modifies, copies, or otherwise does anything which requires a licence in relation to the documentation, they are required to irrevocably undertake to make the source publicly available to anyone to whom they provide an item made to the design (unless the recipient of the item already has access to the design—for example, where the licensee gave it to them personally). The original licensor has the option to add notices to their design which must be preserved when the design is copied, and those notices can include a requirement to put a URL on any item made to the design, or on its casing or packaging, to make it as easy as possible for any recipient of the physical product to have access to the design.

A key innovation in the CERN OHL is the implementation of 'Available Components'. Broadly, the idea is that many designs are likely to consist of a number of components. For an electronic design, this is likely to include commonly available electronic items such resistors, capacitors, and transistors. Where the components are readily available with appropriate interfacing information (in the case of an electronic component, this will commonly be supplied as a data sheet), then there is no need to provide any source for that component. However, if there are any specialist components (perhaps a component such as an inductor which has been specially designed for the project) that are not readily available, then either the source for that component must be supplied under a compatible licence, or sufficient information must otherwise be provided to enable it to be

[39] <http://www.tapr.org/Ackermann_Open_Source_Hardware_Article_2009.pdf> accessed 20 April 2022, and a telephone conversation between John Ackermann and the author.

[40] See section 3.3. <https://ohwr.org/project/cernohl/wikis/Documents/CERN-OHL-version-1.2> accessed 17 June 2022.

made. The logic behind this mechanism is to encourage modular thinking and to avoid the boundary problem. For example, a computer may consist of the components: motherboard, a power supply, a case, connectors, memory, disk drives, etc. Each of those are components, so the computer design can be provided as (at the top level) a case incorporating a number of components, and then (at the next level) each of those components can be either available components or they can be separately licensed under a compatible licence (e.g. a motherboard), and then the components of that item (e.g. the processor on the motherboard) must itself either be an available component, or must itself be licensed, and so on. This also makes it very easy to utilise a part of a design (e.g. the power supply) in another, completely unrelated design: if it is separately licensed, then there is no need to think hard about which components of the Notice file need to be retained, for example.

The distinction between the W (weak) and S (strong) versions is quite subtle and was initially introduced to deal with the rapidly expanding area of open source silicon.

Briefly, open source chip designs are often released as source code written in a hardware description language (HDL), such as Verilog. The source code of HDL looks extremely similar to the source code of a computer program written in a computer programing language like C. The code describes the interconnection of components such as logic gates in the design[41] as well as more complex components. As noted in relation to OpenSPARC earlier, as with a computer program, other components can be imported into the code as 'libraries'. These can potentially be regarded as available components. Under CERN-OHL-S, the provision which allows an available component to be provided without also providing the complete source for it (i.e. if it is readily available with interfacing information) *only* applies to physical components. Accordingly, since these libraries are provided in code form, they are not physical, and therefore the exemption does not apply. The consequence of this is that a designer cannot combine CERN-OHL-S code with other components to make an HDL code design, unless the complete source for those other components is also available, and can also be licensed under CERN-OHL-S. This is similar to the effect of the GPL, which intended to be project-scoped copyleft. In this case, if you are shipping a silicon chip design in HDL and any of the components are licensed under CERN-OHL-S, then you are required to release *all* of the design, including the included components, under CERN-OHL-S[42].

The reality of silicon chip design is that many of the software tools used to design chips are proprietary, and contain their own proprietary libraries and so on. As desirable as it is for proponents of hardware freedom to insist that all components on a chip design are released under a reciprocal (copyleft) licence, in many cases that

[41] Hence the term sometimes applied to HDL designs: *gateware*.

[42] There is also a provision which excludes components included 'as part of the normal distribution of a tool used to design or Make the Product'. Section 1.7(b)(ii).

is just not possible where the toolchain consists of proprietary components, and as a result, a compromise was developed—the CERN-OHL-W.

CERN-OHL-W expands the definition of 'available component' to include non-physical components such as code libraries for silicon chip design. This means that so long as the library is available, and you have information about how to interface it to the rest of the design, then you do *not* have to release the code for the library itself. It is also weaker than the S version in another sense: it is possible to use W-licensed designs as libraries themselves without the reciprocal effect applying to the code in which the library is used. In other words, it is possible to use a CERN-OHL-W licensed non-physical component in conjunction with a chip design which is licensed under a completely different (even a proprietary) licence without all of the code being released under CERN-OHL-W. Any changes to the CERN-OHL-W component will have to be made available to the recipient, and made available under the same licence, but, provided that the rest of the design integrates with the CERN-OHL-W component through its documented interface, there is no requirement to release the rest of the design under that licence. In that way, it's possible to combine CERN-OHL-W-licensed components with components licensed under different (even proprietary) licences.

Those familiar with Open Source software licensing will recognise that this is a similar effect to the one implemented by the LGPL. The distinction is that whereas the LGPL permits the code licensed under it to be linked to proprietary code, it is not possible to incorporate proprietary components within an LGPL-licensed library, the CERN-OHL-W allows both types of combination.

The CERN-OHL 2.0 family of licences also adopts a number of mechanisms similar to those found in Open Source licences: there is a patent licence and retaliation clause similar to Apache 2.0 and a cure period for breach similar to GPLv3.[43]

Although the CERN-OHL 2.0 family is not primarily designed for use with software, it was acknowledged that there may be circumstances in which designers may want to have the whole of a hardware design, and the software used in it e.g. in its firmware) licensed under the same licence.[44] For this reason the drafters decided to submit the three licences to the OSI for official approval as Open Source software licences. Approval was granted in January 2021. The three CERN-OHL v2 licences are the first licences designed primarily for open hardware to be approved by OSI.

[43] See <https://www.ohwr.org/cernohl> accessed 20 April 2022. for more information.

[44] In order to avoid the creation of yet another incompatible commons of software projects licensed under a new reciprocal/copyleft licence, CERN recommends that where software is licensed under the CERN-OHL-W or -S, the licensor should give consideration to dual licensing it under an equivalent software licence, such as LGPLv3 or GPLv3.

23.8 Non-copyleft Hardware Licences

In many cases, it may be more straightforward to use a non-copyleft licence. The Solderpad Hardware License has been developed, based on Apache 2.0 Open Source software licence. The changes revolve mainly around terminology (e.g. expanding the definition of 'source form' to cover more hardware-related forms of documentation), and also the expansion of the types of IP which are covered (database right, for example).[45] The current version acts as a 'wraparound licence' which adds additional permissions and definitions to the underlying Apache 2.0 licence. It also explicitly allows a licensee to regard anything licensed under the Solderpad licence as licensed under plain Apache 2.0. In this way, even though Solderpad is not a licence approved by the OSI (it is not a software licence), anything licensed under it can be regarded as licensed under Apache 2.0 which is an OSI approved licence. Solderpad is, in effect a dual licence: Solderpad and Apache 2.0, at any licensee's option.

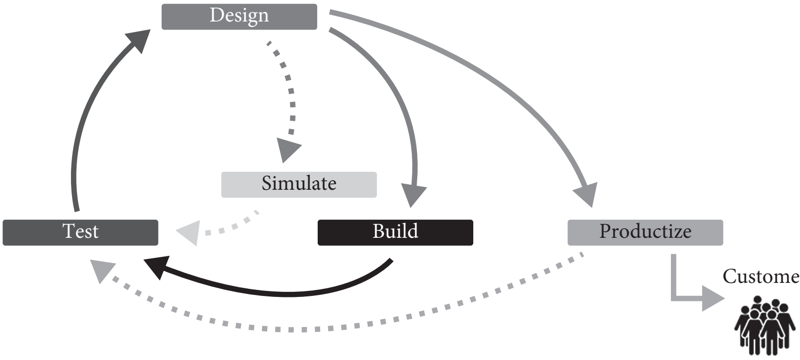
Version 2.0 of the CERN-OHL includes a permissive variant. This removes the reciprocal requirement of CERN-OHL and does not require anyone conveying either a product made in accordance with a CERN-OHL-P-licensed design, or the design incorporating the materials themselves, to be licensed under CERN-OHL-P. It does, however, require the retention of any notices. This is similar in effect to the Apache 2.0 licence, in that it allows free intermingling of code under that licence, and other licences, and distribution of that code under any other licence including a proprietary licence, provided that attributions and notices are retained. As mentioned earlier, CERN-OHL-P is an OSI-approved Open Source licence.

23.9 Open Source Hardware: Development Models

It is generally accepted that one of the drivers to success of Open Source software is that there are very low barriers to entry for participation in many projects. The projects are typically hosted on publicly accessible repositories such as GitLab, and are typically written in languages where the tools—compilers, development environments, and debuggers—are themselves free software. A modest computer, a reliable power supply, and an Internet connection are all that is required to participate. This makes it very easy for individuals to collaborate wherever they are located in the world, and the nature of software means that it is very easy to develop, test and launch a software product as all these activities occur within the virtual digital domain.

[45] <http://solderpad.org/licenses/> accessed 20 April 2022.

The following diagram explains the development cycle:



The core cycle is Design > Build > Test > Design.

In the software world, this typically consists of writing the software (Design), compiling it (Build), and testing it (Test). This process then loops until the software is ready for release, in which case it is made available to the public, maybe by placing the executable in an install package and publishing it on a website, or loading it into a device as firmware which is then shipped. In the world of Open Source, there may be no meaningful productization step, as the software is consistently available as it develops, and where a project has release cycles, it will also update on a much more frequent cadence than proprietary code. Nonetheless, all these activities will take place in the low-cost, low-barrier to entry into the virtual world (except possibly, in a limited number of cases, the productization step). This means the development cycle can be very rapid.

The same core cycle applies to hardware, but it is not necessarily the case that all of it can take place in the low-friction virtual world. In fact, by definition at the very least, the productization and distribution steps must take place in the real, physical world, as may, in practice, other steps.

Increasingly, as a substitute for the build phase, software tools and methodologies including computer-aided design (CAD) and finite element analysis simulation tools can be deployed, enabling much of the development and testing to take place in the virtual world: computers can model things as diverse as earthquake stresses on bridges, the aerodynamic drag of car bodies, the thermal effect of different sorts of insulation on buildings, radio frequency emissions and susceptibility of electronic circuit boards, and the damping effect of suspension components, and therefore the Design > Build > Test cycle can in some cases be made more efficient, cheaper, and faster by substituting it with a Design > Simulate > Test cycle, but even so, there are limitations to the effectiveness of these tools. Another factor is cost: in contrast to the software arena, where some of the finest tools (e.g. the Gnu Compiler Collection) are available for zero cost as Open Source and have become an industry standard, the hardware world lacks an equivalent richness of Open Source tools, and even where the cycle takes place entirely in the digital world, this may require the use of proprietary tools.

This is a potential inhibitor to the efficiency of the open development process as it applies to hardware, and this presupposes that the activities can take place virtually.

Where the development activities have to take place in the real world (e.g. building a dump truck), this can introduce significant expense: it may require a factory, large amounts of energy, expensive capital equipment (lathes, milling machines, materials handling equipment), expensive raw materials (nuts, bolts, steel rods, and steel sheet), physical space and environmental requirements (e.g. a silicon chip foundry will have to be very tightly controlled for dust). And each of these will have to be of a certain and replicable quality. Silicon chips require ultra-pure cylinders of silicon, and even fasteners like nuts and bolts have to be of a certain specification and tensile strength[46]. This also applies to energy: many activities will require a reliable electricity supply without brownouts or spikes. Regulation may be relevant: some activities (making and developing pharmaceuticals and explosives, for example) are very highly regulated. There are even significant regulations applicable to the food and agricultural sectors. This is all much less likely to be the case when dealing with pure software.

All of these things add barriers to entry to the process, inhibiting individuals from entering the cycle, and slowing the cycle itself.

This means that assumptions at the heart of Open Source software and, in particular, the licensing and development models, do not necessarily map well to Open Source hardware, and the extent to which they are likely to map effectively depends very much on how similar the open hardware domain is to software.

As we have seen, hardware description languages are very similar to computer software languages, and can describe complete microprocessor cores. Almost all of the cycle can take place entirely in software, and indeed communities have developed, very similar to Open Source software communities, around different varieties of core (and other associated components). Librecores is one directory of these components.[47] The development of these cores can happen entirely within the digital domain (including simulation and testing for physical characteristics such as thermal requirements, electrical load, and radiofrequency emissivity). It is only at the productisation stage that the core interacts with the physical world, and even then, in many cases, a digital representation of the core (called a 'bitstream') can be created which can be loaded onto a multi-purpose silicon chip called an FPGA. These chips are readily available components, and can cost as little as $5

[46] Incidentally, this highlights another difference between hardware and software. Whereas 'complete corresponding source' in an Open Source licence such as GPL may require the provision of build instructions specifying various configuration files, and the use of certain compilers and linkers within the toolchain, and associated scripts, the extent of similar instructions in hardware may need to be that much greater. Bolts may need to be fastened to a specified torque, fluid levels may need to be measured at a specific temperature, and an analogue radio receiver may need to be set up by adjusting trimmers in a particular sequence.

[47] <https://www.librecores.org> accessed 20 April 2022.

each (or even less in bulk).[48] This means that the development cycle for cores can be very rapid, and the cost of production can be low. In addition, there has been increasing work in the development of open source toolchains, and there now exist a number of open source toolchains which can simulate and synthesise open hardware components such as cores, written in languages like Verilog.

Accordingly, assumptions which apply to the Open Source software development process, and its emergent aspects like community development and business models can also apply to the development of processor cores.

For something like an open source car, these dynamics are much less likely to apply, and open source software assumptions are therefore less likely to apply.

23.10 Conclusion

Open source hardware is a relative newcomer in comparison with Open Source software. As a field it has adopted many assumptions and models from the world of Open Source software, but its latecomer status gives it the advantage of being able to learn from what has gone before.

It is yet to be seen whether the reciprocal (copyleft) model will be as successful in open source hardware as it has been in Open Source software. Its applicability is less straightforward owing to the breadth of IP which can potentially impinge on hardware, coupled with the deceased opportunity for each IP to impinge.

It may well be the case that some types of open source hardware (such as cores) map more closely onto our understanding of the development processes, community dynamics, and business models which are applicable to Open Source software than others.

Open source hardware is a dynamic and interesting field, and is starting to garner significant investment and involvement from industry, and attention from government and policy-makers. In many ways, it will learn from and develop alongside Open Source software but there is still space for it to develop its own direction and dynamics.

[48] The final product may be the FPGA with the relevant core's bitstream installed in it or, if the core is required in much greater volume, a single-purpose chip—an ASIC—can be produced using that core. It is very expensive to manufacture ASICs because it requires preparing semiconductor masks, and setting up costly specialised equipment, but once that has been done, they can be manufactured in bulk very cheaply.

24

Everything Open

Andrew Katz

Openness abounds. Open Source software,[1] is now accompanied by open source hardware[2] (and open hardware),[3] open knowledge,[4] open content,[5] open data,[6]

[1] 'Open Source Initiative' *Open Source Initiative* (1 January 2020) <https://opensource.org> accessed 14 January 2020.

[2] 'Open Source Hardware Association' *Open Source Hardware Association* (17 October 2019) <http://www.oshwa.org> accessed 14 January 2020.

[3] 'What is Open Hardware?' *OpenSource.org* <https://opensource.com/resources/what-open-hardware> accessed 20 January 2020.

[4] 'Open Knowledge Foundation' *Open Knowledge Foundation* <https://okfn.org> accessed 14 January 2020.

[5] 'Defining the 'Open' in Open Content and Open Educational Resources' *Open Content* <http://www.opencontent.org/definition/> accessed 14 January 2020 (hereafter 'Defining the 'Open' in Open Content').

[6] 'The Open Definition' *Open Definition* <http://opendefinition.org/> accessed 14 January 2020.

Andrew Katz, *Everything Open* In: *Open Source Law, Policy and Practice*. Edited by: Amanda Brock, Oxford University Press.
© Andrew Katz 2022. DOI: 10.1093/oso/9780198862345.003.0024

open software services,[7] open politics,[8] open democracy,[9] open government,[10] open public services,[11] open standards,[12] open specifications and formats, open innovation,[13] open education,[14] open publishing,[15] and open access.[16] There is a clearly a connection between these *opens*, but trying to determine the common thread is far from straightforward.

An evident feature of *opens* is that they are intended to remove restrictions to use (including modification and reuse) and access. The terminology may be relatively new, but 'openness' describes an old idea[17]—of sharing and facilitating reuse[18]—which was often the default in the relevant field, until a point in maturity and scale was reached where access became restricted through law or other mechanisms. As a consequence, there was a need to differentiate between that field in its restricted form and the same field made available to all. For example, the free software movement was formed as a reaction to software providers placing restrictions on the use of their software and on access to their source code (following a judicial clarification of the applicability of copyright to code), in contravention of the then default hacker culture of sharing and making the software and its source available.

There is no 'open cuisine' movement, because intellectual property (IP) has not significantly impinged on cooking food or following recipes. This may change if IP rights start to affect these areas such as celebrity chefs seeking to extend the applicability of IP such as copyright or patent to their recipes.

The openness evident in many free software projects goes beyond the ability to access and reuse the code: it also extends to areas like the governance of the

[7] 'The Open Definition' *Open Definition* <http://opendefinition.org/> accessed 14 January 2020.

[8] The Open Politics Manifesto *Open Politics* <https://openpolitics.org.uk/manifesto/> accessed 20 January 2020.

[9] 'openDemocracy' *openDemocracy* <http://www.opendemocracy.net/> accessed 14 January 2020.

[10] 'About Open Government Partnership' *Open Government Partnership*<http://www.opengovpartnership.org/about/> accessed 14 January 2020.

[11] 'Open standards for government data and technology' *GOV.UK* (12 July 2017) <http://standards.data.gov.uk/challenge/open-public-services> accessed 14 January 2020 (hereafter 'Open standards for government data and technology').

[12] 'What Are Open Standards?' *OpenSource.com* <https://opensource.com/resources/what-are-open-standards> accessed 20 January 2020.

[13] 'Open Innovation Community' *Open Innovation Community* <http://www.openinnovation.net/> accessed 14 January 2020.

[14] 'Open education' *Jisc* (31 January 2013) <https://www.jisc.ac.uk/rd/projects/open-education> accessed 14 January 2020, and open educational resources: 'Open Educational Resources (OER)' *UNESCO* (5 June 2019) <http://www.unesco.org/new/en/communication-and-information/access-to-knowledge/open-educational-resources/> accessed 14 January 2020.

[15] Matthew Arnison, 'Open publishing is the same as free software' *Purple Bark* (March 2001) <http://www.purplebark.net/maffew/cat/openpub.html> accessed 14 January 2020.

[16] 'An Introduction to Open Access' *Jisc* (17 October 2019) <https://www.jisc.ac.uk/guides/an-introduction-to-open-access> accessed 20 January 2020.

[17] Charlotte Waelde argues that the first English language scholarly journal, *Philosophical Transactions* (published in 1665) was, in effect, the first open access journal: Charlotte Waelde, 'Scholarly Communications and New Technologies: The Role of Copyright in the Open Access Movement' in Lilian Edwards and Charlotte Waelde (ed), *Law and the Internet* (Oxford: Hart Publishing 2009) 395.

[18] As we will see, there are several movements which seek to define the characteristics of their particular *open*. These definitions are frequently based on the FSF's Four Freedoms or the OSI's openness criteria.

projects themselves (although, as we discuss in Chapter 2, it is by no means necessary for an Open Source project to have an open governance model). Thus the word *open* has established additional shades of meaning and its usage varies from field to field. There are, we suggest, four different but related connotations to the word *open*: use-freedom, open governance (transparency and open participation), and anti lock-in.

24.1 Freedom to Use, Study, Modify, and Share

The most accessible definition of *open* emphasises freedom of use, study, modification, and sharing. The aim of this freedom is to maximise the availability of knowledge and data, and potentially to create a 'commons' of material (in whatever form, including code, content, and data) which is available for use by anyone, for any purpose, with minimal restriction. The restrictions which are considered acceptable are those (such as copyleft), which are intended to ensure that material which is released under an open licence remains open and attribution (which allows a requirement that notices such as copyright notices are preserved when any material is distributed). These freedoms (and the corresponding permitted restrictions) are encapsulated in the Open Knowledge Definition, discussed in more detail shortly. We call it *use-freedom*. A further connotation of use-freedom is that the material must be amenable to study (so anyone can see how it functions), development, and modification for any reason, including extending functionality or field of use, and for correcting errors.

24.1.1 Use-Maximization or Anti-Closure?

Even for proponents of use-freedom there is a tension between those who want to maximize sharing and reuse (irrespective of what the recipients might do with the shared material, and accepting the inclusion of that material into closed products), and those (who tend to emphasise the concept of *freedom*) who want to place obstacles in the way of those who would limit sharing and reuse. These goals are not mutually exclusive and are frequently well aligned, but to understand the dynamics within (and sometimes between) each of the *open* fields, it is necessary to appreciate this distinction.

For example, free software advocates promote the GNU General Public Licence (GPL) family of software licences. By implementing copyleft (see Chapter 3), these licences attempt to ensure that a program subject to the GPL can only be redistributed by a recipient if the recipient also makes any work based on the program subject to the same licence, which in turn becomes binding on downstream recipients, requiring them to comply with the GPL when and if they distribute the program or a work based on it. On the one hand, this fights the perceived evil of 'closure'. On the other, the implementation of copyleft is itself a form of restriction on sharing and reuse. There are examples of software which has not been adopted by a business because it is subject to the GPL, and for the business, copyleft is an unacceptable restriction and may present the business with what it perceives to be unacceptable risks.

In contrast, the ASF releases its software under a liberal licence which imposes little restriction on recipients. Those recipients are able to take the software, adapt it, and redistribute it to third parties (with or without those adaptations), even under a closed, proprietary licence. The foundation's aim is to maximise utilisation and promulgation of the code, even if that means it is capable of becoming closed.

The Free Software Foundation (FSF) is sometimes associated with an anti-closure view; the Open Source Initiative (OSI) with a use-maximization view.[19] The subtle distinction is clear in the world of free and Open Source software, but is less clear in other areas, where the word *open* may be used without distinguishing between the two different (but frequently aligned) concepts. The distinction becomes clearer when looking at the types of licence which are adopted within the various fields.

Use-maximization also requires that the material in question can be used by as many people as possible and for as many purposes as possible, without discrimination, either in relation to characteristics of the individual (or organisation) seeking to use the material, or in relation to the intended use to which the material may be put.[20]

The first meaning of *open*—use-freedom—and the distinction between use-maximization and anti-closure is concerned with the content itself and how it can be used, modified, and distributed. It is not so much concerned with the process of how it came to be created and who can be involved in that process, which is where the next two connotations of *open* apply.

[19] To some degree, these positions are generalisations, as the various organisations have individuals with differing views within them. Having said that, broadly, the FSF will argue that by ever-increasing the pool of GPL software, it will become more difficult for non-GPL software to be developed (given that very little software is developed from scratch these days, but is an assemblage of different components), and that, accordingly, it will maximise the use of free (GPL) software. The Apache Software Foundation (and, to an extent, the OSI) will argue that, although proprietary companies may choose to close Apache (or other permissively licensed) code by releasing it under a restrictive, proprietary licence, they will ultimately realise that it is better business to keep the code open and contribute to the code base, as the benefits of participating in the software's development and support community will lead to a better business outcome. As such, there is a normative effect for code, even under a non-copyleft licence like Apache, to become and remain open. Thus in practice there is significant commonality between the two viewpoints.

[20] This requirement, contained in both the FSF's Four Freedoms and the OSD from the OSI, causes some counterintuitive consequences. For example, both sets of criteria outlaw a licence which only allows non-commercial use, and a notorious seemingly liberal software licence (JSON) which exhorts the licensee to use the software for good, not evil, also falls outside both criteria, as discriminating against those who want to be evil.

24.2 Open Governance

The word *open* carries a second connotation, that of transparency in the governance of a project (in terms of making access to, and records of, the decision-making process freely available), and a related third connotation, the ability of individuals and business to influence and participate in the project on a non-discriminatory basis. Both these connotations are linked by a common requirement that the governance structure allows accountability. We call these two characteristics *transparency* and *open participation*. They are both features of *open governance*.

Transparency is often a characteristic of open development. It is a key characteristic of the Open Source development model described by Eric S Raymond in *The Cathedral and the Bazaar*. According to Raymond, 'release early, release often' (let everyone have access to your development process) because 'given enough eyeballs, all bugs are shallow'.[21] Open Source software projects are often portrayed as meritocracies[22] in which any contributor has an equal opportunity to have his or her efforts recognised in each code release, based solely on the merit of their submissions (see further Chapter 2). A true meritocracy will have a very low barrier for entry and no discrimination both in terms of who can participate and how the participants' contributions are judged. These are prerequisites for open participation.

Open governance may apply to some high-profile projects, but there are also projects in which the code development, although ultimately released under an Open Source licence, is effectively undertaken in a non-open process, carried out by a development team structured in a way very similar to the development teams at any proprietary software company.[23] The terms *open politics* or *open government* imply transparency and open participation.

24.3 Anti-Lock-In

Finally, there may be an assumption that a user of something open is not *locked in*. This means that adoption of the *open*—usually a standard, or a specification, or specific software or hardware—does not create barriers to the adoption

[21] Eric S Raymond, *The Cathedral and the Bazaar* (2 August 2002) <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html> accessed 14 January 2020. The latter is called 'Linus's Law' in honour of Linus Torvalds, initiator and original author of the Linux kernel.

[22] Note that the word 'meritocracy' was coined by Michael Young in his book 'The Rise of the Meritocracy 1870–2033' (London: Thames and Hudson, 1958) and was intended as 'a satire on the folly of meritocratic life' (Ansgar Allen, 'Michael Young's The Rise of the Meritocracy: A Philosophical Critique' (2011) 59(4) *British Journal of Education Studies* 367–382).

[23] Stirling and Bowman describe the distinction, when related to open hardware, as ODH (openly designed hardware) and OWR (open when ready). The distinction applies equally to developments other than hardware. <https://www.tandfonline.com/doi/full/10.1080/14606925.2020.1859168> accessed 20 April 2022.

of another solution, or make it more difficult to move away from an existing solution. For example, the adoption of an open standard for a document format means that the document can be edited, at least in theory, by a number of different word processors, whereas a proprietary format may only be capable of being read by a specific supplier's product. Lock-in frequently manifests itself as a failure of interoperability.

This is the connotation which is particularly relevant in the definition of *open software service*. A further connotation of this is persistence: for a technology to avoid lock-in, its provision must also be persistent (and stable). In other words, the user must be able to trust that the means of access used at a particular time will continue to be available for some significant time into the future. The mirror of this is lock-out which can also be seen in the implementation of incompatible service layers within software as a service provision or cloud environments.

24.4 Interrelationship Between Opens

Free software and Open Source software are, more often than not, the same thing, even though the organisations that are behind them may be perceived to differ in their aims.[24] The development of Open Source (or at least the poster-child projects, like the Linux kernel) often involves open development methodologies, and transparency. Open Source software often complies with Open Standards (which are explained in some detail later in this chapter). Arguably, any interfaces implemented in Open Source software are capable of forming a *de facto* Open Standard,[25] at least potentially, because it is possible to see the code which implements it, and to reuse or re-implement the standard in an open form.

Likewise, it is difficult for Open Source code to be used to implement a solution which still suffers from lock-in as access to the code means that the functionality and characteristics of a solution can be fully understood, and either rehosted using the same code, or re-implemented using different means. (Lock-in in this context

[24] The differences are often exaggerated: Simon Phipps, past president of the OSI, is fond of saying that Open Source is essentially a marketing program for Free Software. See Erlang Solutions, '20 Years of Open Source Erlang: OpenErlang Interview with Simon Phipps' *Erlang Solutions* (19 October 2018) <https://www.erlang-solutions.com/blog/20-years-of-open-source-erlang-the-openerlang-interviews/> accessed 14 January 2020.

[25] Björn Lundell usefully characterises standards as falling on a plane with two orthogonal axes: one axis denotes formality (where a formal standard is one defined by an independent standards body such as the International Organization for Standardization (ISO), and an informal one which would include a standard defined by a commercial organisation), and the other denotes openness, where a maximally open standard is one which complies with all the criteria of (in his specific example) the European Interoperability Framework version 1. See Björn Lundell, Jonas Gamalielsson, and Andrew Katz, 'On Implementation of Open Standards in Software: To What Extent Can ISO Standards be Implemented in Open Source Software?' (2015) 13 *International Journal of Standardization Research* 47.

is difficult, but not impossible, as we shall see in the context of software as a service, and particularly the discussion of an open software service later in the chapter).

There may be conflicts between opens. Open standards are useful only to the extent that they remain stable and consistent. If someone unilaterally 'improves' a standard by amending it, so that devices or software built to that standard no longer interoperate (either at all, or unreliably), then it is no longer of any use as a standard. To be of use, open standards need to be developed in such a way that improvements and amendments are coordinated, and standards bodies have a role here. The pace of development of Open Source code often means that a *de facto* standard like this can emerge much more quickly than a standard which has developed through the standards process. Standards bodies are increasingly recognising this, providing a fast-track procedure for *de facto* standards to be adopted as formal standards. For example, ISO/IEC has adopted a fast-track procedure which can be used within JTC 1 allowing certain members to submit fully formed *de facto* standards into the ISO/IEC standardisation process. One such example is OpenChain (ISO/IEC 5230:2020), discussed further in Chapter 6.

Ideally, the process leading to the formation of the standard itself should be transparent and allow for non-discriminatory participation, but the process and the result are often confused, and to say that it is necessary for both the process and the result to be open (in two quite different meanings of the word) for a standard to be open is unhelpful and an oversimplification.

The outcome of a standards process (whether open or otherwise) should be the creation of a standard which is documented on an open content basis (for which detail, see shortly) and which anyone can implement without payment of royalties or other restriction. This is one definition of an open standard.[26]

Each of the opens we consider in this chapter places slightly different emphasis on each of these characteristics: use-freedom, open governance (transparency and participation), and anti-lock-in.

24.5 Openness and Intellectual Property Rights

We have seen that for a field to develop an open movement there has to be a corresponding closure, or at least a threat of closure. In the case of content (including software code), these threats are largely facilitated by the application of IP.

Where no IP (such as preparation of food to a particular recipe) are applicable, it is more difficult to close the field: a diner in a restaurant who enjoys a particular aubergine curry is free to try to recreate the recipe at home without the permission of the chef. Further, if someone watches the chef prepare the dish on television,

[26] Lundell, Gamalielsson, and Katz, 'On Implementation of Open Standards in Software: To What Extent Can ISO Standards be Implemented in Open Source Software?', see note 25.

they can make notes of the ingredients, quantities, and process, and the viewer is, again, under no restriction regarding creating a recipe accordingly.[27] It follows that there is no 'open recipe' movement, as there are no closed recipes against which it can rally.[28]

That is not to say that where there are no IP, there is no possibility of a corresponding *open* movement developing. Lawrence Lessig has famously drawn a distinction between East Coast code (laws) and West Coast code (physical or technological constraints).[29] Thus, in the absence of laws, it is possible to construct constraints technologically. A simple example is that manufacturers may deliberately make items which are difficult to fix, with the intention of either ensuring that only that manufacturer (or its licensees) has the ability to repair and maintain those items, or that broken items are discarded and a new replacement is purchased.

To repair such items, the owner would ideally have access to blueprints and circuit diagrams; any device would be easy to open using readily available tools rather than employing special security fastenings; replacement components of the device would be readily available and reassembling the device would be possible without special jigs or adhesives. These are the sorts of freedom which are set out in the *Maker's Bill of Rights.*[30]

The EU has recently moved to legislate in this direction, by proposing a right to repair.[31] This is aimed mainly at requiring manufacturers to continue to support devices (including providing electronic updates) and continuing to ensure that commercial repair businesses continue to have the support they need, rather than granting direct rights to consumers, but it can be regarded as a step in the right direction.

[27] Although clearly a written recipe would be subject to copyright as a literary work, making the dish is not infringement of that work (it is interesting to speculate whether a recipe could possibly be considered analogous to 'performing' it like a dance). It may also be the case the dish is associated with a trademark, registered or unregistered, so selling a condiment you have made as McDonald's Special Sauce would invoke a different set of intellectual property rights.

[28] There have been a few attempts to release 'open recipes'. However, it is not clear what those involved are trying to achieve. OpenCola is a recipe for a flavoured sugar syrup intended to be similar to the base for Coca Cola. Coca Cola may famously keep its recipe confidential, but anyone purchasing a bottle of Coke is entitled to reverse-engineer it (a mass spectrometer may help) and publish the results without restriction. The Coca Cola Corporation is unable to exercise any IP to prevent the purchaser from making a clone drink, in the way that it *would* be able to prevent someone from using its name or its distinctively shaped bottle. Open Cola adopts the GPL as its licence, but since making a drink from a recipe does not impinge on copyright in the way that compiling a piece of software does, it's hard to see how any obligations can be placed on downstream recipients of the cola. Reproducing the recipe (i.e. making copies of its text) does impinge on copyright, so it's not so much the cola that is open, than its recipe, in the form of a literary work. See 'OpenCola Softdrink' *ColaWP* (20 February 2001) <http://www.colawp.com/colas/400/cola467_recipe.html> accessed 14 January 2020.

[29] Lawrence Lessig, *Code Version 2.0* (London: Basic Books 2006) 72 et seq. (<http://codev2.cc/download+remix/Lessig-Codev2.pdf>)

[30] Phillip Torrone, 'The Maker's Bill of Rights' *Make* (1 December 2006) <https://makezine.com/2006/12/01/the-makers-bill-of-rights/> accessed 14 January 2020.

[31] <https://www.europarl.europa.eu/news/en/press-room/20201024IPR90101/eu-consumers-should-enjoy-a-right-to-repair-and-enhanced-product-safety> accessed 14 January 2020.

A more subtle application of IP consists in their importance in facilitating copyleft. The copyleft principle provides that if someone makes use of material available under a copyleft licence, then that person will be required, under certain circumstances,[32] to make their amendments, design documents, and/or source code relating to that material available to downstream recipients under the same licence. Sometimes known as *sharealike or reciprocity*, this principle is designed to ensure that once material is available under an open licence, it, and its derivatives, will remain available under that licence. Clearly, however, for copyleft to be effective, there does need to be an IP which would be infringed at the appropriate time, but for the licensee's compliance with a condition in the licence.[33] If the licensee is able to do that act in question without impinging on any IP, then there is no requirement to comply with any licence. This is explored in greater depth later in this chapter.

Notable here is that the evolution of either legal or other constraint is generally driven by the desire for economic value and return (based on the assumption—itself challenged by proponents of openness—that creating artificial barriers or monopolies will always lead to a net benefit for the owner of the monopoly, or controller of the constraint).

24.6 Definitions of Openness (and Freedom) in Software

The most venerable criteria for openness, as currently understood, are the Four Freedoms espoused by the FSF, as discussed in Chapter 3.[34]

These are concerned with use-freedom, and cover use-maximization (and, in a roundabout way, through the words 'access to the source code is a precondition of this' anticipate anti-closure) but the definition does not cover open governance. As we have seen, anti lock-in is an emergent characteristic of Open Source, and the Open Source development model, although applicable to a number of projects, is by no means universal (and not automatically emergent).

Accordingly, it is dangerous for other *opens* to assume that the Four Freedoms can be transmuted into other areas, and the same connotations of openness preserved.

For example, the Open Hardware and Design Alliance proposed a set of freedoms based on the FSF's Four Freedoms, with a reasonable degree of success.[35]

[32] The circumstances will depend on the licence: for example, the GPL is concerned about distribution. The AGPL, additionally, is concerned about access to the functionality of the software across a network.

[33] There have been attempts to create copyleft obligations through a web of contracts, but the issue here is that the scheme fails as soon as any one entity acquires the material in question free of a contractual restriction, which may occur because the entity upstream of them has breached their own contract.

[34] 'What is free software?' *GNU Operating System* (30 July 2019) <http://www.gnu.org/philosophy/free-sw.html> accessed 14 January 2020.

[35] <http://web.archive.org/web/20160624162529/http://www.ohanda.org/> accessed 14 January 2020. OHANDA no longer has a web presence at ohanda.org, and the Wayback Machine last shows a functioning website at that address on 24 June 2016 (accessed 20 January 2020).

Chapter 23 demonstrates that there is an issue with this definition—concerning the words 'complete design'—but subject to that, the definition overall makes sense. However, how can the FSF's definition be meaningfully adopted in a way that makes it clear what 'open government' or 'open politics' is?

24.7 Open Knowledge

The Open Knowledge Foundation (OKF)[36] is a body that has attempted to establish a universal definition for the *opens*. The full open definition is based on the Open Source Definition (OSD) (which is itself based on the Debian Free Software Guidelines).[37] It summarises its own definition (which it calls the Open Definition) as follows (current version 2.1): 'Knowledge is open if anyone is free to access, use, modify, and share it—subject, at most, to measures that preserve provenance and openness.'[38] The reference to 'measures to preserve openness' is probably best explained by reference to an earlier version of the Open Definition (version 1.0): 'A piece of content or data is open if anyone is free to use, reuse, and redistribute it — subject only, at most, to the requirement to attribute and/or share-alike.' It echoes the idea of share-alike but goes somewhat further, in that it encompasses additional restrictions which, for example, limit the ability of the application of digital rights management to restrict recipients of the material from making practical use of it.[39]

Interestingly, although the Open Knowledge Definition as a whole is based on the OSD, the summary above is most closely related to the Four Freedoms and, as such, it covers use freedom but is not concerned with open governance.

Aware that the open knowledge definition is not all-encompassing, the OKF presents another, very specific definition, the Open Software Service Definition: 'A service is open if its source code is Free/Open Source Software and non-personal data is open as in the Open Knowledge Definition.' This cleverly uses the Open Knowledge Definition to define the scope of the data (excluding personal data) which the service contains and processes, and also (indirectly) to define the scope of the software which must be used (Open Source falls within the open knowledge definition). However, this definition is not ideal, in that it still allows for a software service which can be subject to lock-in, as we will discuss.

[36] 'Open Knowledge Foundation', see note 4.

[37] 'Open Definition' *Open Definition, Version 2.*1 <https://opendefinition.org/od/2.1/en/> accessed 14 January 2020 (hereafter 'Open Definition 2.1').

[38] The open knowledge definition is itself published under the Creative Commons Attribution 4.0 International licence, and to comply with that licence in respect of the extracts used, we acknowledge the copyright of the OKF.

[39] This is reinforced by the statement in section 2.2.6 of the definition that '[t]he license may require that distributions of the work remain free of any technical measures that would restrict the exercise of otherwise allowed rights.' See 'Open Definition 2.1', see note 37.

One of the activities of the OKF is, like the FSF or the OSI, to consider which licences are approved, such that material released under such a licence can be regarded as compliant with the open knowledge definition.

24.8 Open Data

The Open Data Commons[40] is a project of the OKF. It seeks, like the Creative Commons organisation discussed later in the chapter, to facilitate the availability of content, in this case data contained in one or more databases, as part of a knowledge commons. The tools used are a set of licences: the Public Domain Dedication and Licence, the Attribution Licence, and the Open Database Licence.[41] These licences are drafted to take into account the special characteristics of databases, such as the EU's *sui generis* database right, and the practical consequence of a multi-source database potentially containing contributions from many thousands of different sources (at which point it becomes impractical to provide attribution for them all).[42]

Another organisation, the Open Data Foundation,[43] focuses not just on making data available but on making it useful by seeking to promote global metadata standards (metadata is data about data), so that information from diverse databases conforming to the standards can be combined in practical and interesting ways. As such, its activities are intended to help people:

Discover the existence of data
Access the data for research and analysis
Find detailed information describing the data and its production processes
Access the data sources and collection instruments from which and with which the data was collected, compiled, and aggregated
Effectively communicate with the agencies involved in the production, storage, distribution of the data
Share knowledge with other users

[40] 'Open Data Commons' *Open Data Commons* <http://opendatacommons.org/> accessed 14 January 2020.

[41] 'Licenses' *Open Data Commons* <http://opendatacommons.org/licenses/> accessed 14 January 2020.

[42] 'Licenses FAQ' *Open Data Commons* <http://opendatacommons.org/faq/licenses/> accessed 14 January 2020. Richard Poynder, 'Interview with Jordan Hatcher' *Open and Shut?* (18 October 2010) <http://poynder.blogspot.co.uk/2010/10/interview-with-jordan-hatcher.html> accessed 14 January 2020. Jordan Hatcher is the principal drafter of the Open Data Commons licences.

[43] 'Open Data Foundation' *Open Data Foundation* <http://www.opendatafoundation.org> accessed 14 January 2020.

24.9 Open Content

Open content is another blanket term and is broadly equivalent to 'open knowledge'. At least terminologically, the movement is roughly as old as the OSI (the terms *open source* and *open content* were both coined in 1998). Initially intended to refer to content licensed under the Open Content Licence, the definition extended to any content meeting the following criteria:[44]

Reuse—the right to reuse the content in its unaltered/verbatim form (e.g. make a backup copy of the content)
Revise—the right to adapt, adjust, modify, or alter the content itself (e.g. translate the content into another language)
Remix—the right to combine the original or revised content with other content to create something new (e.g. incorporate the content into a mashup)
Redistribute—the right to share copies of the original content, your revisions, or your remixes with others (e.g. give a copy of the content to a friend)

These criteria are very similar to the FSF's Four Freedoms, so it will come as no surprise that the Open Content Licence[45] is modelled on the GNU GPL, in that it is a copyleft licence. It does restrict the ability to charge for access (and to that extent is partially non-commercial).[46]

24.10 Creative Commons

One of the most prominent organisations in the open content movement has been Creative Commons. The brainchild of Professor Lawrence Lessig, with co-founders Hal Abelson and Eric Eldred, the name *Creative Commons* draws on the metaphor of creative activity being what economists call a *commons*. However, whereas a physical commons (e.g. fish in the sea, or a piece of common land in a village) can be exhausted by harvesting or overgrazing, a commons of ideas cannot be exhausted, as using an idea does not remove it from the commons (in economic parlance, use of material subject to intellectual property is non-rivalrous).

Creative Commons is intended to cover a wide range of material, such as literary works, photography, video and film materials, and other works such as choreography, with the intention that as many works as possible are available for reuse.

[44] 'Defining the 'Open' in Open Content', see note 5.

[45] <https://web.archive.org/web/19981206111937/http://www.opencontent.org/opl.shtml> accessed 20 January 2020. The Open Content Licence (and its sister licence, the Open Publication Licence), have now been deprecated in favour of the Creative Commons licences.

[46] Non-commercial licences cannot be free or Open Source licences because these licences must permit commercial use. This is explored in greater depth later in the chapter.

Creative Commons consists of a series of licences, which are currently on their fourth version. They have been localised for use in a number of jurisdictions worldwide to address differences in copyright and other laws between jurisdictions. A content owner choosing to use a Creative Commons licence has a number of options: BY (attribution), ND (no derivatives), NC (non-commercial), SA (share alike). These options can be combined in various combinations. Thus a licence designated CC-BY-NC would allow the user to take the work and exploit it for non-commercial purposes (NC), provided that the originator is credited (BY). Permission is also granted to amend the work. The ND (no derivatives) variant prevents derivative works from being made. SA (share alike) is a copyleft-like provision that requires any re-distribution of the work or a derivative to be subject to the same licence.

Creative commons licences have gained wide acceptance: for example, the whole of Wikipedia is released under CC-BY-SA. The use of simple tags to designate the applicable licence has the benefit of making material that is available under a specific licence to be easily identified and searched using a search engine such as Google. Sites such as flickr.com have added functionality to make it easy to identify material which is available under particular licences.[47]

Works such as software or hardware are occasionally released under Creative Commons licences. This is not appropriate, as the licences do not deal effectively with the distinction between (for software) source code and object code or (for hardware) the design documents, and the hardware itself, and they do not cover patents. (If the material relates solely to a design document rather than the hardware instantiation of the design, then a Creative Commons licence may be appropriate.)

On the other hand, where the material is more analogous to computer software (e.g. multi-track source material for use in a digital-audio workstation such as Reason[48] or Ableton[49]), then a licence like the GPL may be more appropriate. However, this is unusual: it is normally unlikely that an Open Source licence will be an appropriate choice for material other than software.

Material is sometimes released under ‘a Creative Commons licence’ without further qualification: this is unhelpful. Creative Commons is a suite of licences, and without further information it is not possible to determine which of the suite is intended.

Despite the *open* credentials, not all Creative Commons licences comply with the various free/open criteria. For example, the non-commercial (NC) option discriminates against commercial fields of endeavour. (It is also difficult to determine exactly what ‘non-commercial’ means; at one extreme it could be taken to limit use

[47] ‘Explore / Creative Commons’ *Flickr* <http://www.flickr.com/creativecommons/> accessed 14 January 2020.

[48] ‘Reason Studios’ *Reason Studios* <https://reasonstudios.com> accessed 14 January 2020.

[49] ‘Music production with Live and Push’ *Ableton* <https://www.ableton.com/en/> accessed 14 January 2020.

of material for which direct payment is taken, and at the other, it could be taken to mean use in any context which involves an organisation which receives money.)[50] Likewise, an ND (no derivatives) version of the licence prevents amendments being made to the content, and also fails to provide use-freedom.

The NC Creative Commons licence is also the source of a great deal of confusion from those without a detailed understanding of the nuance of licensing and the OSD. It is not uncommon for even knowledgeable developers and lawyers to seek and frequently base decisions on a belief that there exists an NC Open Source licence. This is, of course, not possible, as we have seen, since commercial activity is a field of use, and restriction of a field of use brings a licence outside the OSD.

Creative Commons has also suggested a mechanism for dedicating works to the public domain[51] called CC0. Because dedication to the public domain is not possible in many jurisdictions, including England and Wales and in Scotland, CC0 includes a section entitled 'Public Licence Fallback' which grants the widest licence possible should dedication fail.[52]

24.11 Other Documentation Licences

A number of other open licences are available for documentation, the best known of which is probably the GNU Free Documentation Licence.[53] The FDL is a part copyleft licence which contains some complex terminology intended to ensure that software documentation, in particular, remains free but useful and relevant to the software which it documents.

24.12 Open Hardware (and Open Source Hardware)

There have been several attempts to apply open principles to physical objects. The step from Open Source to physical hardware seems at first sight to be fairly straightforward. However, there are several issues which make this more complex.

The term 'hardware' covers a wide range of things, from mechanical items like cars and pillar drills, to electronic items like computer printed circuit boards, to aesthetic items such as statues or friezes: indeed, it can mean any physical but

[50] The terminology used in the most recent versions of the licences, which states that non-commercial 'means not primarily intended for or directed towards commercial advantage or monetary compensation' does not assist greatly.

[51] 'Our Public Domain Tools' *Creative Commons* <http://creativecommons.org/publicdomain/> accessed 14 January 2020.

[52] 'CC0 1.0 Universal' *Creative Commons* <http://creativecommons.org/publicdomain/zero/1.0/legalcode> accessed 14 January 2020.

[53] 'GNU Free Documentation Licence' *GNU Operating System* (3 November 2008) <http://www.gnu.org/copyleft/fdl.html> accessed 14 January 2020.

non-solid objects, including liquids like beer or cleaning fluid. Even within the field of electronics, there is a significant distinction between a printed circuit board which is clearly a physical object, and individual components such as FPGAs (field programable gate arrays), the most important characteristic of which is that their operation is determined by a stream of digital code, which is in many ways akin to software.

The legal basis for the use, modification, and distribution of content, including software, is generally better understood than it is for the use, modification, and distribution of hardware, and the types of IP which apply to hardware are more diverse (and also tend to vary more significantly from jurisdiction to jurisdiction) than those which apply to software.

An effective open hardware licence will either limit itself to particular forms of hardware or will be aware that hardware can come in this wide variety of forms.

There are far fewer open hardware licences than there are Open Source licences as the field of open hardware is less developed than that of free and open source software.

These issues are covered more comprehensively in Chapter 23.

24.13 Open Data

Open data is a form of open content and arises in many contexts. Proponents of open government are keen to see governmental statistics (and their underlying datasets) being made freely available, and indeed freedom of information legislation is a useful tool to facilitate this.[54] In academia, projects such as genetic research and nuclear physics have generated vast amounts of data which, it has been argued, should be made freely available to facilitate research. There are also specific initiatives, such as OpenStreetMap, which are intended to facilitate crowdsourcing geographical data, and open genealogy, covering family history.[55]

In the US, the government has launched data.gov,[56] which is a central repository of governmental data. The UK has data.gov.uk.[57] The release of governmental data fulfils two requirements, each characterised by a slightly different connotation of *open*. The first is that if data is made freely available for reuse, then the availability of the data will reduce friction in the rest of the economy, and promote the

[54] In the UK, Freedom of Information Act 2000, Freedom of Information (Scotland) Act 2000 and Environmental Information Regulations 2004. The Aarhus Convention (1998) also provides for public access to certain information: 'Public Participation' *UNECE* <http://www.unece.org/env/pp/welcome.html> accessed 14 January 2020.

[55] 'Manifesto: The Right to Culture in the Digital Age' *Open Genealogy Alliance* <http://www.opengenalliance.org/> accessed 14 January 2020.

[56] 'The home of the U.S. Government's open data' data.gov <http://www.data.gov> accessed 14 January 2020.

[57] 'Find open data' *data.gov.uk* <http://data.gov.uk/> accessed 14 January 2020.

development of useful applications for data which will themselves stimulate the economy and provide a benefit to everyone. The second is more allied with the concept of *open governance* and is intended to promote democracy by providing the transparency necessary to hold government to account. From this perspective, the availability of the data is more akin to access to information through mechanisms such as freedom of information requests.

The rationale behind non-governmental projects like OpenStreetMap is focused on the first, use-and-access connotation of openness, rather than any form of transparency.

24.13.1 Data: the legal and licensing context

The US has historically had neither a *sui generis* database protection right, nor protection for databases as an extension of copyright.[58] Further, in the US, copyright protection is not available to works created by the government,[59] and such works are assumed to be in the public domain, the theory being that if tax dollars have been expended in creating them, they should be free to use by anyone (it has been assumed that this includes anyone outside the US, but this idea has been challenged, including on the US government's own website).[60]

The situation in the EU, in particular, is very different. The EU possesses a *sui generis* database protection right.[61] In addition, the EU Directive on the reuse of public sector information (2003/98/EC)[62] has as its rationale that copyright works owned by the government should be exploited by licensing them to commerce for the highest bidder. So, for example, mapping data which in the US has historically been available universally as it is in the public domain, is considered public sector information in the UK, use of which is made available under a restricted licence through the Ordnance Survey.[63] Despite these philosophical issues, much information is now made available through the data.gov.uk portal under less restrictive licensing. The data available through data.gov.uk is available not only as dumps

[58] *Feist Publications, Inc., v Rural Telephone Service Co.*, 499 US 340 [1991]. Various copyright protections may be available at state, as opposed to Federal, level, but these are beyond the scope of this chapter.

[59] US Code, Section 105, Title 17, Chapter 1.

[60] US government, 'U.S. Government Works' *USA.gov* (18 July 2019) <https://www.usa.gov/government-works> accessed 14 January 2020.

[61] Directive 96/9/EC of the European Parliament and of the Council on the legal protection of databases [1996] OJ L77/20. The EU Database Right has been subsumed into the UK domestic law after Brexit, although the extent of mutual recognition has been limited: <https://www.gov.uk/guidance/sui-generis-database-rights-after-the-transition-period> accessed 14 January 2020.

[62] Directive 2003/98/EC of the European Parliament and of the Council on the re-use of public sector information [2003] OJ L345 as amended by Directive 2013/37/EU of the European Parliament and of the Council [2013].

[63] <https://www.ordnancesurvey.co.uk/business-government/licensing-agreements> accessed 20 January 2020.

of data in various database formats (the simplest being comma-separated values (CSV)), but access is also available through an application programing interface (API).

The licence under which much of the data on data.gov.uk is released is the Open Government Licence, which is a liberal database licence, requiring only attribution.[64] It does limit its own scope to personal data (and it is unclear whether the definition of *personal data* employed is the fairly restrictive definition contained with the Data Protection Act 2018[65] which deals solely with data relating to a living individual, or a wider interpretation which covers any data relating to any individual, irrespective of whether the individual is living or dead).[66]

The US site data.gov does not specify a licence, possibly because of the twin assumptions that (i) data as such is not amenable to IP protection under US law, and (ii) that governmental data would, in any event, be in the public domain.

Both availability of the data and the ability to reuse it under a liberal licensing framework are important to advocates of openness, but as services are built on top of government data portals, the reliability and persistence of the API are also important. In addition, where people wish to compare data between portals from different jurisdictions, it becomes increasingly necessary that those portals adhere to standards, so that to extract equivalent data from different portals it is not necessary to customise the interface code each time. Thus the additional connotation of anti-lock-in—persistence—is critical in this context. To address these issues, the OKF has developed CKAN, an Open Source data management system[67] which is intended to act as a platform to facilitate data transfer, in part by using standardised data catalogues which facilitate the comparison of data within and between datasets.

There are several different mechanisms to facilitate interoperability between datasets. One of these is the *semantic web* project, which aims to provide a way of categorising data to create what it calls a *web of data*. In this way, data presented by different entities from different datasets is provided in a predictable way to facilitate programatic access to web sites and the data behind them. The Semantic Web Project[68] is led by the World Wide Web Consortium (W3C). The Open Data Foundation also seeks to facilitate interoperability and combination of data sources.[69]

[64] 'Open Government Licence for public sector information' *The National Archives* <http://www.nationalarchives.gov.uk/doc/open%20government-licence/> accessed 14 January 2020.

[65] <http://www.legislation.gov.uk/ukpga/2018/12/part/2/chapter/1/enacted> (accessed 7 January 2020), which was passed in response to Regulation (EU) 2016/679 of the European Parliament and of the Council on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation) [2016] OJ L119 (GDPR).

[66] GDPR, art 4(1).

[67] 'CKAN, the world's leading Open Source data portal platform' *CKAN* <http://ckan.org/> accessed 14 January 2020.

[68] 'W3C Semantic Web Activity' *W3C* <http://www.w3.org/2001/sw/> accessed 14 January 2020.

[69] 'Projects' *The Open Data Foundation* <http://www.opendatafoundation.org/?lvl1=projects> accessed 14 January 2020

The Open Data Institute (ODI), founded by web inventor Tim Berners-Lee and Nigel Shadbolt, also seeks to promote access to government data.[70]

24.14 Open Software Services

The OKF promotes a definition of Open Software Services.[71] The core definition is:

An open software service is one:

1. Whose data is open as defined by the Open Definition with the exception that where the data is personal in nature the data need only be made available to the user (i.e. the owner of that account).
2. Whose source code is:
   1. Free/Open Source Software (that is available under a licence in the OSI or FSF approved list . . .).
   2. Made available to the users of the service.

The Open Definition is a significant part of this definition and goes some way to extending the definition away from use-freedom and helping to deal with lock-in. However, a better way to look at openness in software services is that they must also minimise lock-in (and, ideally, be based on standards which are open, and therefore also require open governance) which also leads to lock-out of competitors.

Lock-in can impinge in two ways: the interface presented by the API must be open, in the standards sense of it being fully documented, and available for use without payment of any royalty.[72] Further, we argue that it must also provide appropriate and complete functionality on a non-discriminatory basis. 'Appropriate and complete' means that that functionality must do everything that the user would need it to, including facilitating bulk extraction of data (including metadata and, where necessary, pending transactions) at any time during the lifecycle of the solution. 'Non-discriminatory' means that each user has access to the full API, and that the API performs in the same way for each user (so that there are no hidden API

[70] 'About the ODI' *Open Data Institute* <http://www.theodi.org/about> accessed 14 January 2020.

[71] 'Open Software Service Definition' *Open Definition* (8 October 2008) <http://opendefinition.org/software-service/> accessed 14 January 2020.

[72] This does present a problem: there must be some mechanism to enable to the providers of open software services to charge for their services. One logical way to gauge whether charging is reasonable or not is to look at the model of the GPL, which does not prohibit or limit charging for software (except in the specific case of providing a copy of the source code in response to a request), but relies on economics to lower the cost of the software to the marginal cost of copying it, on the basis that any licensee with a copy of the source code is able to copy and distribute it without any fee to the licensor, so competition between licensees will tend to drive the cost of copies down to the marginal cost of copying it. Likewise, if the software providing the open software service is available on a Open Source basis, then it would be open to any person with a copy to instantiate their own competing service, which would tend to drive the cost of providing the service down to the marginal cost.

calls for privileged users), and that the performance is not dependent on which user is accessing the API—a vendor, for example, might seek to lock in one customer by only allowing a certain number of API calls per second, preventing bulk transfer of data.

Transparency is related to the documentation of the API but it requires a further step: that the mechanisms underlying the provision of the service are also fully documented, in a form that would enable them to be reproducible (this is similar to one distinction between *open hardware* and *open source hardware* which is discussed in Chapter 23). Of course, providing the source code of all of the software providing the service would fulfil this requirement (at least assuming that there is no non-software element of the service which is also required to provide the service).[73] However, even without access to the underlying source code, if the API and the mechanism for delivery of the service are available, then a competing service can be provided.

It is relatively easy to imagine a service which fulfils the requirements of the Open Software Services Definition, but which still fails to be *open* in any practical sense because the data which it contains are not capable of being effectively extracted. It may be because the software does not contain appropriate functionality to allow the data to be extracted.[74]

The Open Software Services definition, in its FAQ, does go some way in addressing these points, by stating: 'The Open Definition also requires that data should be accessible in some machine automatable manner (e.g. through a standardized open API or via download from a standard specified location).' But this does not handle a situation where there is discrimination in the access available through the API. It also incompletely addresses the issue of metadata.

An appropriate definition of an open software service must, therefore, address the issues of transparency and lock-in. The user of a service described as open must be able to transfer to another service with minimal effort:[75] the data must, both theoretically and practically, be portable.

[73] It was rumoured that one provider of mobile speech-to-text services performed a large proportion of its work by using human labour offshore, rather than computers. There is no requirement for the services provided on the back side of an API to be provided by computers.

[74] Just because the software is released as Open Source does not mean that the user of the service will have any ability to change the version used in that instance of the service to rectify problems with the API's functionality, although they may be able to create their own instance and modify the code to deal with those issues. However, the new instance is unlikely to be of any use unless the data accessed through the original instance is accessible.

[75] Rufus Pollock, who happens to be on the board of the OKF, makes a relevant point about lock-in as it applies to virtual worlds. Virtual worlds are different from the real world, in that it is relatively straightforward to up sticks and move from one virtual world to another. In the real world, moving from one country to another can be enormously disruptive. As complex as it may be to move from World of Warcraft to Second Life, it's an order of magnitude easier than moving countries in the physical world. In the physical world, we need democracy to exercise control over the governance. In the virtual world, a governor who fails to provide an amenable environment for the subjects will find all those subjects leaving to a more conducive domain. Governors will, of course, be aware that failing to provide for the

Practical portability means that it must also be possible to extract all of the user's data, including metadata, from the service in a meaningful, sensible way. There are two ways in which access to data can be restricted: by law (or by terms of use restricting certain activity in relation to API, such as number of calls per second) and by code (there is a hard-coded restriction on the number of API calls per second, for example). Both of these must be absent for a software service to be properly regarded as open.

Therefore, a better definition of an open software service will be one which takes into account not only anti-closure and use maximization but also transparency and, crucially, anti lock-in. Anti lock-in also implies that a competing service vendor must not be locked out of the service layer of a cloud provider's platform.

24.15 Open Politics and Open Government

Open politics and open government are about transparency and trust or accountability. In addition, open politics and government principles require the information and data used to make decisions to be made freely available to as wide a group as possible, and accordingly, use maximising becomes relevant. The mechanism by which people are able to participate in the democratic process should also be non-discriminatory.

The UK government has published a white paper which defines open public services[76] as an important branch of open government. Open public services are those which display the following characteristics, summarised as:

> Choice and control—Wherever possible the government will increase choice.
> Decentralisation—Power should be decentralised to the lowest appropriate level.
> Diversity—Public services should be open to a range of providers...
> Fair access—the government will ensure fair access to public services.
> Accountability—Public services should be accountable to users and taxpayers.

Where information or data are made available under an open government basis, it is important that they are capable of broad reuse. See section 24.8 for more information about how this is typically carried out.

needs of their subjects will cause them to lose subjects. Accordingly, even if they are dictators, they will still have a normative pressure imposed on them to behave well and keep their subject happy. The same applies so far as software services are concerned: if it is relatively easy to transfer from one software service to another, then the provider of the software service will work that much harder to provide a service which its users are happy with.

[76] 'Open standards for government data and technology', see note 11.

24.16 Open Standards and Open Specifications

Standards exist to promote interoperability (see further Chapter 12). As such they need to be applied consistently. Any standard which is subject to the ability to adapt, amend and extend in an uncontrolled or arbitrary fashion rapidly loses this consistency, and becomes untrustworthy, and ultimately valueless. Accordingly, the application of the description *open* to a standard is necessarily subtly different from its application in other contexts. This does mean that an open standard may be subject to restrictions that are not necessarily applicable to other opens, in order to ensure consistency.

There are several different definitions of open standard.

The first version of the European Interoperability Framework contained the following criteria:[77]

24.16.1 Use of open standards

To attain interoperability in the context of pan-European eGovernment services, guidance needs to focus on open standards. The following are the minimal characteristics that a specification and its attendant documents must have in order to be considered an open standard:

- The standard is adopted and will be maintained by a not-for-profit organisation, and its ongoing development occurs on the basis of an open decision-making procedure available to all interested parties (consensus or majority decision etc.).
- The standard has been published and the standard specification document is available either freely or at a nominal charge. It must be permissible to all to copy, distribute and use it for no fee or at a nominal fee.
- The intellectual property—i.e. patents possibly present—of (parts of) the standard is made irrevocably available on a RF basis.
- There are no constraints on the re-use of the standard.

This requires use-freedom not only of the standard itself (bullets 3 and 4), but also of the documentation comprising the standard (bullet 2). Additionally, open governance of the standards creation process is dealt with in bullet 1, as is anti-lock-in (in that the standard cannot be amended without an appropriate decision making

[77] European Communities, *European Interoperability Framework for Pan-European eGovernment Services* (1st edn, Brussels: Office for Official Publications of the European Communities 2004) (available at <https://op.europa.eu/en/publication-detail/-/publication/a4778634-27fa-43b4-9912-f753c4fdfc3f>) accessed on 21-Sep-22.

process open to all interested parties, so any user of the standard is less likely to be subject to arbitrary and unplanned changes).

One contentious issue which this definition directly addresses is the requirement for an open standard to be usable without payment to any holders of IP (in particular, patent holders). Clearly, if a standard becomes adopted, this is likely to significantly increase its use.

A patent holder whose patent would necessarily be infringed by implementation of the standard is potentially in a position to demand significant sums for licensing the patent. This sometimes happens when the patent holder is part of the team establishing the standard in the first place (in which case the patent holder can either be open about holding the patents from the outset, as was the case with Sony and Phillips when they established the various CD and CD-ROM formats, or the patent holder can keep the existence of the patents or the applications for them secret).[78] It can also happen when an entity is not involved in the standards-setting process but begins to assert its patents after the standard has been put into effect.

If a standard is an open standard (under the EIF version 1 definition, for example), then a licence to the patents which would be infringed by its implementation must be available on a RF basis to anyone. It is an open question what happens if patents are later discovered which impinge on a standard which was previously understood to be open, either because they belong to a party who participated in the standards setting process and who concealed the existence of the patents, or because patents belonging to a third party are subsequently discovered which impinge on implementations of the standard. It can be assumed that if the patent holder is unwilling to make licences to the patents irrevocably available on a RF basis to all implementers of the standard, then the standard can no longer be described as 'open'. However, this can have consequences: what happens, for example, if a procurement process is already underway which specifies that the items to be procured must comply with an open standard? What if some items have already been procured in the belief that the standard is open, the standard then becomes closed, and further items need to be procured which, to interoperate with the already-procured initial items, now need to comply with a standard which is no longer open?

A further issue is that, even if licences to the patents are available on a royalty-free basis, those patents are likely to impose other terms on the licensee, and those terms may be equally problematic to proponents of openness, and particularly

[78] This is known as a 'patent ambush'. Rambus was alleged to have done this, by participating in setting standards for RAM through the standards-setting body JEDEC. After the standards were ratified, it was alleged that Rambus started to demand royalties from entities using the particular technologies covered by its patents and that Rambus kept details of the patents secret from the standards body and the other participants, an allegation which Rambus denied. The facts of the case are complex, and impinge on competition law both in the US and the EU. A good summary of the issues from a European perspective can be found here: <ec.europa.eu/competition/publications/cpn/2010_1_11.pdf> accessed 20 January 2020.

those who would wish to implement the standard in free or Open Source software, as we explore shortly.

24.16.2 FRAND

Patent holders may argue that it is a step too far to demand that they make a licence to their patent available to all implementers of the standard on RF terms. They may offer to make the patents available under 'reasonable and non-discriminatory' terms ('RAND') or 'fair, reasonable and non-discriminatory' terms ('FRAND').

RAND and FRAND have been dismissed as being meaningless platitudes by proponents of openness[79] in that the interpretation of what is fair and reasonable is too subjective, and that if royalties are payable then the licences will necessarily discriminate against Open Source. This is explored more thoroughly below.

A further variety of patent licence, FRAND-Z, adds a requirement that zero royalties are payable, which does render the licence compatible with EIF1 requirements but does not address interaction between other aspects of the licence and certain Open Source licences. This is particularly problematic in relation to the GPL, as will now be discussed.

24.16.3 Interaction between Open Source and open standards

The GNU GPL family of licences (from GPL version 2 onwards) contains the 'liberty or death' clause discussed in Chapter 3 which, although it varies slightly from one version of the licence to another, provides that if a licensee cannot license the covered code to a third party so that the third party gets the same rights that the licensee has, then the licensee may no longer distribute the covered code. This is intended to prevent a scenario where a licensee obtains a personal licence (to a patent, for example) which enables it to use the covered code, but any downstream licensee cannot benefit from that licence, and must approach the patent licensor for another licence (which the licensor may charge for), to allow the software to continue to be used. This was seen as an unwarranted restriction on freedom (and would permit the implementation of a relatively simple mechanism to be employed allowing a patent licensor to subvert software freedom—something that the FSF, with its anti-closed stance, was unwilling to countenance).

In order to be compliant with the liberty or death clause, any patent licence which impinges on a GPL program must allow the same patent licence to be made

[79] Simon Phipps, 'Why RAND is bad for Open Source' *Computerworld* (20 April 2012)<https://www.computerworld.com/article/3423951/why-rand-is-bad-for-Open Source.html> accessed 20 January 2020.

available to any downstream recipient of the code. In particular, this means that the patent licence needs to be sub-licensable to any downstream recipient of the GPL code, or that at the very least that a patent licence is available to any downstream recipient of the code. Whilst is it possible that a FRAND licence could be constructed which complies with the GPL requirement, this is unlikely, as such a licence would have to be so wide[80] that it would be equivalent to a patent surrender (in other words, giving up the patent entirely).

In practice, FRAND licences are not sub-licensable. Even if the FRAND licences are available to downstream recipients, they are likely to be conditioned on the downstream recipient implementing the standard, and not deviating from it. The GPL, however, requires that a downstream recipient must be free to modify the code, even if the modification means that the code deviates from the standard. (This is an illustration of the conflict between *open* principles in that use-maximisation, in the guise of allowing modifications, conflicts with the anti-lock-in principle that needs standards to be maintained). The original licensee of the GPL code cannot comply with its obligation to allow downstream recipients to modify the code,[81] the liberty or death requirement is not satisfied, and the GPL code cannot therefore be distributed.[82] It has also been argued that the cascade licensing model of GPL itself (i.e. that each downstream recipient of GPL code receives a number of licences, one from each contributor to the code received) is incompatible with the direct licensing model that is present in almost all FRAND licensing structures.[83]

Thus even FRAND-Z is incompatible with a GPL licensing model, and FRAND more so.

Open standards are covered more comprehensively in Chapter 12.

24.17 Open Innovation

'Open innovation' is a term coined by Henry Chesbrough in his book *Open Innovation: The New Imperative for Creating and Profiting from Technology.*[84] It

[80] It is possible to imagine a FRAND-Z licensing structure broad enough to be compatible with the GPL licensing model—it would have both to allow sub-licensing, and also to allow implementations other than those which implement the standard.

[81] This right does not have to extend to the right to allow the downstream recipients to modify the software in such a way that it might infringe a different patent which was not licensed under the original FRAND licence.

[82] A counter-argument is that the original recipient of the GPL code (assuming it takes the benefit of the same FRAND licence as are available to downstream recipients) is limited in the same way as any downstream recipient, and that therefore, since the original recipient has no greater rights than are granted to any downstream recipient, there is no loss of freedom, and the liberty or death clause does not impinge.

[83] Iain G Mitchell QC, Stephen Mason, instructed by Andrew Katz, 'Compatibility of the Licensing of Embedded Patents with Open Source Licensing Terms' (2011) 3 *Journal of Law, Technology and Society*, available at <http://www.ifosslr.org/ifosslr/article/view/57/99> and <https://www.jolts.world/index.php/jolts/article/view/57/99> (both accessed 7 January 2020).

[84] Harvard Business School Publishing Corporation, 2003.

differs from the other *opens* in that it does not envisage universal access to knowledge and the use of innovative IP licensing structures or governance to facilitate access Instead, and more restrictively, it aims to make organisations more amenable to the inventions and innovations from outside the organisation, and to allow organisations to license their IP to others more freely. This is likely to result in a wider dissemination of invention and innovation, but only under a traditional network of non-disclosure agreements and licences. Accordingly, *open innovation* is not concerned with the forms of openness considered in this chapter.

24.18 Open Publishing, Open Education, and Open Access

Open publishing and open access mainly concern academic articles, journals, and publications such as this book and refer to the ability of a researcher to access the journal or article without payment, and, ideally, to reproduce the relevant text also without any restriction (including payment).[85] There are several competing definitions, but possibly the most succinct is that developed by the US Public Library of Science (PLoS): 'Free availability and unrestricted use.' Open access publishing is not incompatible with traditional 'consumer-pays' publishing, and open access journals can still provide high-quality peer-reviewed material. Since electronic publication and distribution is inexpensive and effective, almost all, if not all, open access journals are available online.[86] Many have print versions available as well. The print versions may be paid for by advertising, sponsorship, an article processing charge paid for by the author (or the author's sponsor or affiliated organisation) or by a charge to the purchaser (the fact that an open access journal is available at a price in print form does not preclude it from being an open access journal, so long as it is possible to obtain a version, usually electronic, without charge). Someone downloading a copy of an open access journal can, of course, also print one or more copies of the journal, or use a print on demand service like Lulu.com to create a physical copy.

The website doaj.org (Directory of Open Access Journals) lists 14,182 journals qualifying as open access journals[87]

In practice, many open access journal articles are published under an appropriate Creative Commons licence, although other licences, like the free documentation licence, are also employed. The choice of licence is important, and there is often an assumption that any Creative Commons licence will do. This is incorrect, and most importantly, the NC (non-commercial variants) and ND (no derivatives)

[85] Waelde, 'Scholarly Communications and New Technologies', see note 17.

[86] For example, the *Journal of Law, Technology and Society* (formerly *International Free and Open Source Software Law Review*) at <https://jolts.world>.

[87] As at 14 January 2020.

are not appropriate. See, for example, the RCUK Guidance[88] which stipulates that a CC-BY licence should be used, although a CC-BY-NC licence is acceptable in some limited circumstances. CC-BY-ND licences are never appropriate for an open access publication.

Open access is divided into *green* and *gold* standards.[89] The chief difference is not in terms of licensing but that the green standard refers simply to the information being made available but without a formal mechanism for review. The gold standard adds a requirement for the information to be peer-reviewed, and is thus competitive (and seeks to be as authoritative as) academic peer-reviewed journals published under the traditional mechanism.[90]

The main *open* addressed in open access publishing is use-freedom.

'Open education' has a connotation that is similar, and refers mainly to courseware being made available on an open access basis (also referred to as open courseware).[91]

However, the meaning may be much broader, and refer to elimination of barriers restricting access to education. A prime example is the Open University[92] which opened in the UK in 1971. Open education providers rely heavily on (and contribute to and create) open courseware, but also provide more traditional services provided by open education bodies: tutorials, access to academic staff, examinations, etc. The Open University (OU) requires no academic qualifications for entry (although qualifications are required to move on to more advanced courses, which can be obtained within the OU itself, so that a sufficiently dedicated and able student can enter the OU with no qualifications and leave with a doctorate, or even post-doctoral qualification). The OU does charge fees (although grants may be available to qualifying students).

Other open education institutions operate on a similar basis. Use-maximization is the goal, although the governance of open-education institutions may also involve transparency. Open education institutions are often keen to ensure that their academic qualifications are regarded as equivalent to those from non-open institutions, and, accordingly, will submit to review by bodies, often governmental, which

[88] <https://www.ukri.org/wp-content/uploads/2020/10/UKRI-020920-OpenAccessPolicy.pdf> accessed 20 January 2020.

[89] An Introduction to Open Access' *Jisc* (17 October 2019) <https://www.jisc.ac.uk/guides/an-introduction-to-open access> accessed 20 January 2020.

[90] For example, the *Journal of Open Law, Technology and Society* (<https://jolts.world>) (formerly the *International Free and Open Source Software Law Review*) is now published under a Creative Commons Licence, generally CC-BY. Its licensing model has gone through some iterations (with the intention always being that its content was to be freely available). Because its academic articles are peer reviewed, they meet the gold standard of open access (it does carry opinion articles which are less rigorously reviewed, and these, despite being licensed on the same basis, meet green open access standard).

[91] 'Open Education Consortium' *Open Education Consortium* <http://www.ocwconsortium.org/> accessed 14 January 2020.

[92] 'The Open University' *The Open University* (2020) <http://www.open.ac.uk/> accessed 14 January 2020.

are established to ensure that standards are maintained. An example in the UK is the Quality Assurance Agency (QAA).[93] This also has the effect of reducing lock-in by ensuring that students have the ability to move between accredited institutions throughout their academic lives.

24.19 Summary

An increasing number of fields describe themselves as '*open*'. The term most frequently connotes access to and the ability to reuse data and information, but, within those connotations, there is a tension between maximising the use of content, and preventing it from being re-closed (use maximisation, and anti-closure).

This is not all. Transparency in governance (including access to and influence of the decision-Open-source software and making process) is frequently connoted by the term '*open*', as is a rejection of structures which would allow dominance by a specific entity or group of entities: 'lock-in'.

The four connotations can be described as use-freedom, transparency, open participation, and anti lock-in. An understanding of these four different connotations assists in determining the commonality between the many different *open* fields.

[93] <https://www.qaa.ac.uk> accessed 18 February 2021.

Appendix

Issue	Commentary	Who is best placed to bear risk?	Best mechanism to tackle risk
Supplier-created code infringes copyright	The risk of detection of infringement is easier for [F/OSS] (as the code is more readily available for comparison purposes, especially if the code is GPL and re-distributed, but the ability of the customer to mitigate its loss is greater, as it automatically has access to the source code, to enable it to re- engineer infringing code itself if the Supplier will not or cannot do so.	Supplier	Indemnity/warranty from Supplier. Supplier has right to rewrite infringing code. Version control system (VCS) shared repository and allowing audit rights
Publicly-available code (i.e. code acquired from third parties under a [F/OSS] licence, and incorporated into the software) infringes third party copyright.	One risk is that the publicly available code selected is inherently infringing (i.e. there is a provenance issue), or alternatively, the component is available under a [F/OSS] licence, but not the one attached to it. Note that "publicly available code" will include publicly available code that has been amended by the Supplier to create a derivative work.	Varies from project to project. If the Customer specifies use of a specific component, then it should be liable for claims in relation to that component. If the Supplier selects the components, there is a stronger argument that the Supplier should bear some of the risk, or at least take care in the selection process.	Warranty or indemnity from the Supplier, to encourage Supplier to take care in source selection. A list of agreed sources of code may give the Customer comfort (even if this is by no means conclusive), and may encourage the Supplier to take fewer risks in terms of provenance. Further, if code is obtained from recognised locations, it is more likely to be heavily reused, and therefore there is, arguably, safety in numbers (i.e. the code has been used many times before and there hasn't been a claim yet), and also the likelihood that if it is found to be infringing, the community will generate a non-infringing alternative

(*continued*)

Issue	Commentary	Who is best placed to bear risk?	Best mechanism to tackle risk
	Another risk is that the Customer (as opposed to the Supplier) may specify the use of specific [F/OSS] components, and in using these components faces a similar issue as above, though with a different context for allocating potential liability.	If the Supplier selects the components, there is a stronger argument that the Supplier should bear some of the risk, or at least take care in the selection process. If the Customer performs this selection, the opposite is true.	Customer takes all risks relating to the nominated code.
	It is possible to explicitly address the risk of publicly available code not being available under the licence apparently attached to it, and instead actually falling under a different licence and potentially incompatible licence.	This is similar to the provenance issue, in that the Customer's use/modification/distribution of the Software may infringe third party rights, but in this case, infringement may depend on the Customer's intended out-licence or intended use of the Software. This wording contains an option which limits the Supplier's obligations to checking that the components' attached licences are on an approved list, but not that they are compatible with any intended use.	Warranty relating to the licences attached to publicly-available code components. Optional exclusion of liability for licence incompatibility (Customer takes risk of incompatibility).

	Sweeper up warranty designed to ensure that code-selection for copyrights is within the ambit of the Supplier's services.	Supplier	Warranty that skill and care has been taken in component selection, so far as third party copyrights are concerned
	Publicly available code is incompatible with the Customer's Specified Use or Specified Out- Licence. By requiring the Customer to specify in this way, expectations are managed, and minds are focused. Note that 'Specified Use' may include internal use only with no possibility of distribution (which allows code to be mingled freely), or internal use only with interaction over a network externally, (which allows code to be mingled freely, unless network-aware licences like AGPL or OSL are employed).	Supplier (in relation to code and module assembly) and Customer (in ensuring that the code is only used for the Specified Use)	
Infringement by misuse of third party code by the Customer.		Customer	

(*continued*)

Sample Wording	Supplier's Arguments	Customer's arguments	Comments
The Supplier warrants that it has title to all Supplier-Created Code and that its delivery [assignment/licence] to the Customer and use in accordance with this Agreement does not infringe the [copyright] of any third party.	No good ones	Supplier is in control of code creation, and should therefore be liable for third party infringements. Supplier should use a common source code repository, to which Customer may be given access.	Note that supplier-created code may in practice amount to amendments to existing publicly available code (and create a derivative work of that code). In that case, the row below (publicly available code infringes copyright) would be more appropriately apply)
The Supplier warrants that each component of Publicly Available Code incorporated in the Software has been acquired solely from the locations listed in Appendix [1] and that the source of each such acquisition shall be accurately documented [as set out in Appendix [2]]. [The Supplier further confirms that it has compiled, [with reasonable skill and care], documentation required by the licence[s] applicable to the Publicly Available Code and will provide it to the Customer in order to enable the Customer to comply with [the notice and disclaimer] conditions applicable to such licence[s]]	Each Customer has a different appetite for risk. Requiring the Customer to document how it regards the risk of accessing code from different locations, gives the Supplier more information on which to base an accurate price for the job. Alternatively, Supplier may want to give the Customer the option of a cheaper price by doing "quick and dirty" development by scraping code from anywhere, without provenance checking, providing that the Customer takes the risk. In any case,	Supplier is contracting to supply IPR, and should bear all the risk. How Supplier intends to source IPR should not be Customer's issue. In any event, where the Supplier is actively choosing the code to use, provenance checking should be a selection criterion.	Infringement can occur either because the infringing code is not available under any [F/OSS] licence (e.g. it is derived from proprietary code), or because it is not available under the licence supposedly attached to it (e.g. it is available under the GPL, but appears to be available under the BSD). It may also occur becausethe code has not been released by complying with the ap

	this clause as drafted could prove unduly restrictive for the Supplier. There are vast amounts of quality code available from “grey” sites. Also, is “reasonable skill and care” capable of consistent interpretation given the state of the art? Koders.com contains plenty of roll-your own licences, for example. Also, just because something is on sourceforge.net does not mean that it is necessarily of any better provenance than elsewhere.	propriate notices in the licences (e.g. copyright notices, disclaimers). These are reasonably easy to correct.
The Customer acknowledges, notwithstanding any other provision of this Agreement, that the Supplier shall not be responsible for any claim, cost or expense howsoever arising from the Supplier's incorporation, use of, modification of, linking to the Customer's Specified Components [and the Customer shall indemnify the Supplier for any cost, claim or expense arising therefrom].	The Supplier's choice of component is restricted, and therefore it should not be held liable for such use.	

(*continued*)

Sample Wording	Supplier's Arguments	Customer's arguments	Comments
[The Supplier warrants that[, so far as it is aware,] each component of Publicly Available Code incorporated in the Software is available under one of the licences specified in Appendix [3] and has documented the provenance of each such component [as set out in Appendix [1]][The Supplier does not warrant that use, modification or distribution by the Customer of the Software will not infringe the rights of any third party, and no provision of this Agreement or implied term shall be construed as such a warranty].	The Supplier does not want to be responsible for ensuring licence compatibility, as the Customer will be much better placed to determine what its intended use is. Therefore, it's more practical for the Customer to specify a list of compatible licences, than having the Supplier do compatibility checks.	The Customer selects code	
[The Supplier warrants that it has taken reasonable skill and care in selecting publicly available components having regard to the non-infringement of third party copyrights [the Customer's Specified Use and the Customer's Specified Out-Licence], and has documented the provenance and licences applicable to such components [as set out in Appendix [1] and [2] [with reference to Appendix [3] where applicable]].]	This warranty is too vague, at least without qualification as to whether the licences which are attached to the components are compatible with the Customer's Specified Use or (preferably) the Customer's Specified Out- Licence.	The Supplier needs to be put under a practical obligation to make copyright compatibility/ awareness part of its selection criteria.	

The Supplier warrants that [so far as it is aware, but without having made any specific enquiry] the development of the Software, its delivery to the Customer and the Customer's modification, distribution and use of the Software within the Specified Use [or relicensing to third parties within the Specified Out-Licence]. shall not infringe the licences set out in Appendix [3]. The Customer acknowledges that use of the Software outside the scope of the Specified Use [and any distribution of the Software to a third party] may infringe third party rights.	This warranty places the onus on the Supplier (at least without the awareness qualification) to ensure compatibility, which can include a legal analysis of different licences. This may be outside the scope of the ability of the Supplier, or the scope of the services intended to be provided.	The Customer has taken time to specify either the licences to be used, or the Specified Use, and it is up to the Supplier to ensure that the Software complies with this requirement.	Code with incompatible licences can be intermingled, if there is no question of it being distributed (and, where relevant licences like AGPL and OSL are concerned, accessed externally over a network). If the Customer brief requires this, then it is sensible to include a clause in the agreement stating that the Supplier is acting as the agent of the Customer in authoring the code, to prevent the distribution of code from Supplier to Customer being a distribution. This is particularly important in jurisdictions (like the UK) with no work-for-hire doctrine. It is sensible also to consider present transfer of future rights. See 'Status of Supplier' below.

(*continued*)

Sample Wording	Supplier's Arguments	Customer's arguments	Comments
[The Customer is responsible for ensuring that its own subsequent use, modification and re-distribution of the software [outside the Specified Use] is in accordance with [the licences set out in Appendix [3]].	The Supplier is developing for the Customer. Therefore the Supplier is not to be concerned about out-licensing, outside the scope of the specified use. This is the Customer's issue. Any future or different uses would be subject to a future or different agreement.	The Customer may want to distribute in the future, and may want to out-license to customers etc. Also, passing around the group, or to the acquirer of the business may be "distribution" and therefore should be covered.	
Bought-in proprietary code infringes third party copyright	A third party proprietary library may have a provenance issue – i.e. it obtains code which the supplier is not entitled to license This code may itself be proprietary, or it may be [F/OSS]	Supplier (through contractual relationship with provider of the proprietary code) (unless use of that component is nominated by the Customer – see above)	Indemnity/warranty from Supplier - but can Supplier obtain a back to back indemnity from the provider of that code? This chain may need to extend all the way up to the ultimate provider.
Infringement of patent in Supplier Created Code		Where Supplier has choice of implementation: Supplier. Where implementation is dictated by Customer's requirements: Customer	Right to change implementation, if implementation is determined by Supplier. Otherwise, risk is on Customer. May be possible to negotiate risk sharing. May be possible to get insurance? Audit rights?

Infringement of patent in publicly available code	Where Supplier has choice of implementation: Supplier. Where implementation is dictated by Customer's requirements: Customer	Where implementation is dictated by Customer: Customer to bear risk. Otherwise, negotiated on a case by case basis.
Infringement of patent in bought-in proprietary code	Where Supplier has choice of implementation: Supplier. Where implementation is dictated by Customer's requirements: Customer	Where implementation is dictated by Customer: Customer to bear risk. Otherwise, negotiated on a case by case basis. Can Supplier obtain a back to back indemnity from the proprietary Supplier?
Trade secrets	Supplier	
Trademarks	Customer	
General Indemnity Wording		
Implied terms, pre-contractual representations		
Conduct of Claim		
Access to VCS repository		

(*continued*)

Sample Wording	Supplier's Arguments	Customer’s arguments	Comments
The Supplier [confirms that the licences under which the third party components of the Software are available [are contained within the list set out in Appendix [3] as amended from time to time by agreement between the parties]][will not be breached by the Customer's Specified Use][,permit the Customer to out-license the Software under the Specified Out- License] and that so far as it is aware [but not having made specific enquiry] the development of the Software and its delivery to the Customer do not infringe such licences. [The Customer is responsible for ensuring that its own subsequent use, modification and re-distribution of the software [outside the Specified Use] is in accordance with such licences.][The Supplier agrees to provide reasonable assistance to the Customer in passing the benefit of any warranties associated with such third party [proprietary] components to the Customer subject to the Customer's continued compliance with the licences applicable to such code.	Supplier to use reasonable skill and care in selecting code, but should not be liable for third party infringement. Similar to the supply of third party hardware. May offer to pass on any third party warranties available. May also be subject to the Customer complying with terms passed through by the Supplier.	Supplier is contracting to supply IPR, and should bear all the risk. How Supplier intends to source IPR should not be Customer's issue.	The third party code may infringe because it contains copyleft code, but the source is not made available. This may cause the customer to be the subject of, for example, a GPL violations claim, even if the customer wishes all the code to be available under the GPL, because it does not have access to the source. The customer's remedy may therefore be to compel the supplier to release the source (which may have to pass this requirement up the supply chain). This also suggests a circumstance where the customer insists on the source code being placed in escrow, and released if there is a third party GPL violations claim.

The Supplier warrants that [so far as the Supplier is aware [not having made any enquiry]] the use by the Customer of the Software for its Specified Use [within [jurisdictions]] will not infringe any right which any third party may hold under any valid patent.	It is not economically feasible to undertake a patent clearance prior to implementation. If the implementation is dictated by the Customer's requirements, this should not affect liability. The existence of patents in the Customer's field of business (particularly of they are business-method patents) are part of the cost and risk of doing business in that sector, and the customer should be aware of, and take the risk, accordingly.	Supplier is contracting to supply IPR, and should bear all the risk. How Supplier intends to source IPR should not be Customer's issue.
<none>	It is not economically feasible to undertake a patent clearance prior to implementation. If the implementation is dictated by the Customer's requirements, this should not affect liability. If supplier has to accept some liability for patent infringement, Again there is the potential to insure against this in the UK at a high price and the additional costs of this would be passed through to the Customer.	Supplier is contracting to supply IPR, and should bear all the risk. How Supplier intends to source IPR should not be Customer's issue.

(*continued*)

Sample Wording	Supplier's Arguments	Customer’s arguments	Comments
<none>	Supplier to use reasonable skill and care in selecting code, but should not be liable for third party infringement. Similar to the supply of third party hardware. May offer to pass on any third party warranties available, or to assist and again this may be subject to a pass through of third party restrictions.	Supplier is contracting to supply IPR, and should bear all the risk. How Supplier intends to source IPR should not be Customer's issue.	
The Supplier warrants that, to the best of the Supplier's knowledge [but not having made any specific enquiry], its delivery [assignment/licence] to the Customer and use in accordance with this Agreement does not breach any obligations of confidentiality to a third party.			
For the avoidance of doubt nothing in this Agreement [except for clause []] is intended to grant any licence over any trade mark of the Supplier or its licensors. The Customer shall comply with the terms of the licences governing all third-party components comprised in the Software, which may include terms relating to trade marks.	The Customer may wish to use the Supplier's trade mark if the code is distributed (or accessed remotely). The parties may rely on trade mark law to tackle this, or incorporate an explicit licence permitting the use of the trade mark in relation to the Supplier's code only if it is not modified in any way.		

The Supplier will indemnify and hold the Customer harmless on demand against any claim or loss arising as a consequence of a breach of any of the [above warranties – warranties set out in this clause].

Except as expressly set out in this Agreement, the Supplier makes no representations or warranties in respect of or in connection with the Software or its use. All other representations, warranties, conditions or other terms which might have effect between the parties or be implied or incorporated into this Agreement or any collateral contract, whether by virtue of statute, common law or otherwise, are hereby excluded to the maximum extent permitted by law, including, without limitation, implied conditions, warranties or other terms as to satisfactory quality, merchantability, fitness for purpose or the use of reasonable skill and care.

The Customer shall notify the Supplier promptly (“a Claim Notice”) should it receive any claim that any portion of the code delivered under this Agreement infringes the rights of any third party, or where it otherwise has reason to believe that it does so. The Supplier's obligation to indemnify the Customer under [clause [ ]] in connection with a claim against the Customer by a third party is subject to: (a) the Customer promptly serving a Claim Notice; (b) the Customer not making any admission as to liability or compromising or agreeing to any settlement of any such claim without the prior written consent of the Supplier[, which consent shall not be unreasonably withheld or delayed]; (c) at the Supplier's written request and at its own expense, the Supplier having the conduct of and the right to settle all negotiations and litigation arising from such claim; and (d) at the Supplier’s request and expense, the Customer giving the Supplier all reasonable assistance in connection with such negotiations and litigation. [The Customer shall take all reasonable steps to mitigate its loss arising from any default of the Supplier]

The Supplier undertakes that it will [during the Term] allow the Customer [read-only] access to the [VCS Repository].

(*continued*)

Sample Wording	Supplier's Arguments	Customer's arguments	Comments
Replace or Re-write			
Licence of Collective Work	The Software is likely to consist of a number of components, and the list of components itself will amount to a collective work. Although in many jurisdictions, the collective work will be implied, in some jurisdictions, e.g. Spain, it may need to be explicitly granted. Note also that the GPL may not be an appropriate licence for a collective work – FDL, or creative commons may be more appropriate as they do not introduce source code complications.		
Limitations and exclusions of liability			
Status of Supplier	This needs to be considered carefully in the context of each licence. Generally, the Supplier will want to be providing services to the Customer, rather than deliverables. This has issues for distribution, acquired rights directive, liability.		
Failure of software to meet specification: Supplier created	Note that the source is automatically available. No need for escrow. More natural to have documentation available.	Supplier	Warranty from Supplier + ability to re-write non-performing code

Failure of software to meet specification: publicly available	Supplier, generally	Warranty (negotiated) from Supplier + ability to re-write non-performing code
Failure of software to meet specification: proprietary	Original supplier - can supplier pass on warranties etc?	Back to back warranty from supplier, or mechanism to enable customer to benefit from original suppliers' warranties (agency, third party beneficiary, collateral warranty)

The Supplier may at any time replace any part of the code ("the Original Portion") delivered under this Agreement where it reasonably believes that such code infringes the rights of any third party or where a claim of such infringement has been made, provided that such replacement code materially complies with the Specification. The Supplier shall cease to be liable to the Customer for any claim relating to the Original Portion to the extent that it arises after delivery of the Replacement Code, except where such claims apply to items already created or manufactured and currently being deployed to market.

The Supplier acknowledges that the combination of the components within the Software constitutes a collective work. The Supplier hereby grants a non- exclusive licence to such collective work to the Licensee [consistent with the rest of this Agreement][consistent with the Specified Use]

(*continued*)

Sample Wording	Supplier's Arguments	Customer’s arguments	Comments
The Supplier's liability under or in connection with this Agreement (whether in contract, tort (including negligence) or otherwise) is limited as follows: (a) the Supplier will have no liability for any loss of profits, loss of business, loss of goodwill, loss of anticipated savings, loss of or corruption to data or for any indirect or consequential loss or damage; and (b) the maximum aggregate amount of any such liability which is not excluded by (a) shall be [ ]. Nothing in this Agreement shall limit the Supplier's liability for death or personal injury or arising as a result of fraud.	On a risk and reward basis the Supplier will wish to limit to the fees for the specific project.		
The Supplier is [an independent contractor][acts as Agent for the Customer in developing the Software]			
To the extent that any Supplier-Created Code fails to meet the Specification, the Supplier shall during the Warranty Period [replace such Supplier Created Code with code that is compliant][insert SLA]	Offer SLA? Maintenance agreement. Warranty period. Source is automatically available	Warranty that Software will perform to specification.	

To the extent that any Publicly-Available Code fails to meet the Specification, the Supplier shall during the Warranty Period [replace such Publicly Available Code with code that is compliant][insert SLA]	The Supplier should not be responsible for the performance of third party code.	The Customer should not be concerned about how the Supplier opts to select code. Further, for Publicly-Available Code, the Supplier has access to the source, and can therefore treat that code as simply a more-rapidly-developed version of its own code. There is therefore no reason why it cannot give a warranty.
The Supplier shall take reasonable steps to assist the Customer with the enforcement of any warranties applicable to proprietary code, but shall [except to the extent that no reasonable supplier could have specified the use of such code] not otherwise be liable for any failure of any third party code to reach Specification.	Industry standard to use third party code. Depends on type of code (OS/database engine/library/Embedded component)	Software should perform to specification.

Index

For the benefit of digital users, indexed terms that span two pages (e.g., 52–53) may, on occasion, appear on only one of those pages.