EuropaInstitut

AN DER UNIVERSITÄT ZÜRICH

Sarah Leins-Zurmuehle

Ideation, Conceptualization, Realization

Discovering the Creative Scope in Software Engineering from the Perspective of Copyright and Patent Law

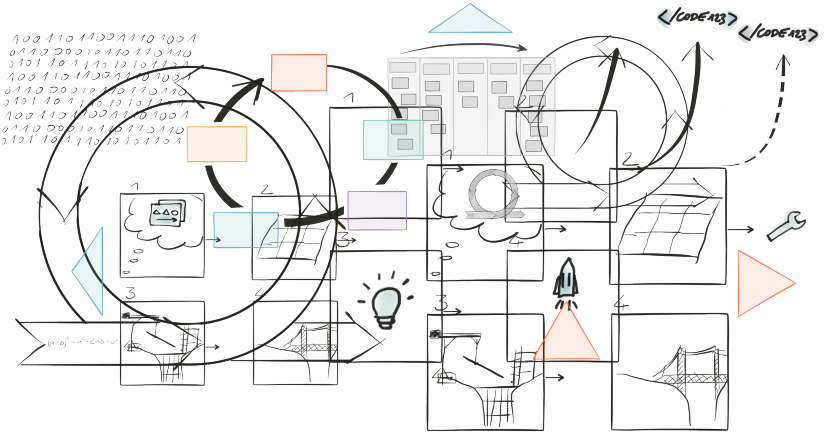
EIZ Publishing

Ideation, Conceptualization, Realization –

Discovering the Creative Scope in Software Engineering from the Perspective of Copyright and Patent Law

Dissertation at the Faculty of Law of the University of Zurich

to obtain a Doctoral degree



submitted by:
MLaw Sarah Leins-Zurmuehle
of Weggis LU

supervised by:
Prof. Dr. iur. Florent Thouvenin
and
Prof. Dr. iur. Andreas Heinemann

Author: Sarah Leins-Zurmuehle
Publisher: EIZ Publishing
Production, Set & Distribution: buch & netz (buchundnetz.com)
Illustrations: © Sarah Leins-Zurmuehle, Robin Leins
ISBN:
978-3-03805-409-2 (Print – Softcover)
978-3-03805-412-2 (Print – Hardcover)
978-3-03805-444-3 (PDF)
978-3-03805-445-0 (ePub)
DOI: https://doi.org/10.36862/eiz-409
Version: 1.05 – 20211008

The dissertation was submitted by Sarah Leins-Zurmuehle and adopted by the Faculty of Law of the University of Zurich, represented by Dean Prof. Dr. Brigitte Tag, on May 27, 2020.
It was supervised by Prof. Dr. iur. Florent Thouvenin and Prof. Dr. iur. Andreas Heinemann.

The open access publication of this book has been published with the support of the Swiss National Science Foundation (SNF).

This work is available in print and various digital formats in **OpenAccess**. Additional information is available at: https://eizpublishing.com/publikationen/ideation-conceptualization-realization/.

"In the end nobody knows how it's done – how art is made. It can't be explained. (...) Understanding a tool doesn't explain the magic of creation. Nothing can." *

* *Quote by David Hockney, an English artist, issued in an interview with Martin Gayford of the Telegraph, published on September 22, 2001, available at <https://www.telegraph.co.uk/culture/4725696/Hockney-and-the-secrets-of-the-Old-Masters.html> (retrieved September 6, 2021).*

Acknowledgements

This book is the result of research for my doctoral thesis at the University of Zurich.

In the course of my dissertation project, I benefitted from the support and advice of many researchers and practitioners:

First of all, I would like to thank my academic supervisors and Ph.D. advisors *Prof. Dr. iur. Florent Thouvenin* and *Prof. Dr. iur. Andreas Heinemann*. Prof. Dr. iur. Florent Thouvenin's sharp way of thinking and cross-linked view challenged me to improve this thesis to its final extent. He generously shared his knowledge and helped me to focus on the important questions. Prof. Dr. iur. Andreas Heinemann has supported my research since the beginning, although at first not in an official role. His inspiring, always constructive, helpful, calm and professional nature allowed me to find confidence in my research question.

Another person without whose work my thesis would not have reached its desired form is *Dr. phil. Claudia Vorheyer*, former Senior Assistant at the Institute of Sociology, University of Zurich. She supported me tenaciously throughout my interdisciplinary research and showed me where to turn. Her infinite knowledge and know-how of sociological methodology is inspiring and her enthusiasm contagious. Thank you, Claudia, for your patience and sincere support. I know you are going to be a wonderful professor!

I would also like to thank the *Europa Institute at the University of Zurich (EIZ)* for making this publication possible and particularly *Atty. Dr. iur. Tobias Baumgartner, LL.M.*, and *MLaw Michael Mayer* for collaborating in an exceptionally supportive and constructive manner, enabling me to finalize and publish this thesis.

On the way, I met so many practitioners who were very supportive. I want to express my admiration for the lawyers, project managers and developers working in this field in an inventive and creative manner. My grateful thanks go out to all the (anonymized) interview candidates and companies who gave me their time and voluntarily shared their knowledge. Further thanks go to those who gave me initial input and helped me to connect with the right people. My particular thanks go to *PD Dr. sc. Matthias Stuermer*, *dipl. HTL. Ing. Frans van der Reijden*, *Atty. Dr. iur. Georg Rauber*, *Atty. Dr. iur. Rolf auf der Maur*, *Atty. Dr.*

iur. Thomas Steiner, *LL.M.*, and *Atty. Dr. iur. Martin Eckert*. I want to say a special thank you to *Atty. Dr. iur. Wolfgang Straub*, *LL.M.*, who supported me with his extraordinary legal and technical expertise in the software field. This exchange and his feedback on my research were vital for the final version. Thank you so much, Wolfgang, for your exceptional support and commitment!

Prof. Dr. iur. Urs Gasser, *LL.M.*, enabled me to spend two inspiring and instructive months as a visiting researcher at the *Berkman Klein Center for Internet & Society* of *Harvard University*. It was in Cambridge where my interview study was finalized. Thank you so much, Urs and the representatives of the Berkman Klein Center, for enabling my stay and making it worthwhile. The Berkman Klein Center represents an accumulation of the most fascinating and enthused researchers and practitioners in the field.

A researcher cannot face the challenges of the research without the support of their closest ones – thank you to my dear friends and family. My special thanks go out to *MSc dipl. Inf. Felix Kaiser* and *Prof. Dr. phil. Stefan Leins*, who spent hours helping to edit my work. I owe you my deepest gratitude. I am greatly indebted to *Atty. Dr. iur. des. Angelika Murer*, my dear friend, with whom I had untiring discussions about her, my and our common research.

Finally, the greatest thanks go to *MA ETH HPK Robin Leins*. His advice was prudent and challenging at the same time, his wonderful sketches were a presentation to the point. Thank you for keeping me focused and smiling, even through the cloudy days.

Zurich, September 2021.

Index

List of Abbreviations XVI

References XXI

I. Bibliography XXI
II. Electronic Sources XLII
III. Others XLVII

List of Figures LI

Abstract LIII

Chapter 1: Introduction 1

I. Introduction to the Research Subject 1
II. Definition of the Research Problem 5
III. Chapter Overview 9

Chapter 2: Methodology 12

I. Introduction to the Socio-Scientific Approach and Selection of a Research Method 12
II. Literature Review 16
III. Interview Series 18
    A. Purpose of the Interview Series 19
    B. Focus of the Interview 20
    C. Selecting an Interview Type 21
    D. The 'Expert' Term 23
    E. Sampling 26
        1. Initial and Theoretical Sampling 27
        2. The Sample Group 29
    F. Preparing the Interviews 32
        1. Anticipating the Interview Situation 32
        2. Structure and Content of the Interviews 34
IV. Transcription 37
V. Data Analysis 38
    A. Selecting a Method 38
    B. Analysis and Theory-Building 41

Chapter 3: Technical Foundation 47
I. Definition of Relevant Terms 47
A. The Science of Software Engineering 48
B. Elements of a Computer Program 54
C. Typology of Computer Programs 59
II. The Development Process 60
A. The Standard Phase Model 61
1. Ideation 64
2. Conceptualization 65
3. Realization 66
4. Implementation 67
5. Review and Maintenance 67
B. Three Different Approaches for Developing Software 68
1. Linear Development 68
2. Spiral Development 71
3. Continuous Delivery 76
III. Software Project Management and Commercialization 79
A. Brief Overview of the Economics of Software Engineering 79
B. Software Project Management 81
C. Commercialization 84
1. Classic Software Commercialization 84
2. Open Source and Free Software in Particular 85

Chapter 4: Status Quo of Legal Software Protection 88
I. The Legal Institutions for Software Protection 88
A. Introduction to Legal Software Protection 88
B. Brief Overview of the Available Legal Protection Measures 89
1. Intellectual Property Law 90
a) Patent Law 90
b) Copyright 90
c) Industrial Design Rights 90
d) Trademark 93
e) Utility Models (Gebrauchsmuster) 95
2. Unfair Competition Law 96
a) Trade Secrets 97
b) Taking Undue Advantage of Somebody Else's Achievement 98
3. Contract Law 100
C. Tabular Summary 101

II. Function of Intellectual Property Law 103
III. International Context 106
A. Paris Convention for the Protection of Industrial Property 107
B. Revised Berne Convention for the Protection of Literary and Artistic Works 108
C. Universal Copyright Convention 110
D. European Patent Convention 111
E. Agreement on Trade-Related Aspects of Intellectual Property Rights 112
F. WIPO Copyright Treaty 113
IV. Patent Law 114
A. Patent as an Intellectual Property Right 115
B. Patent Scope 118
1. Subject Matter 118
a) Interpretation under the European Patent Convention 122
b) Interpretation under the Swiss Patent Code 132
c) Interpretation under the U.S. Code and Affiliated Case Law 134
d) Conclusion: Are Computer Programs Patentable? 137
2. Protection Requirements 138
a) Novelty 138
b) Non-Obviousness 139
c) Industrial Applicability 143
3. Term of Protection 145
a) Starting Point of Protection: Registration 146
b) End of Protection 148
V. Copyright 149
A. Copyright as an Intellectual Property Right 149
B. Copyright Scope 151
1. Subject Matter 152
a) Creative and Artistic Works 152
b) Exclusion of Ideas and Functional Prerequisites 154
aa) Ideas 154
bb) Functionalities 157
c) Current Interpretation of the Subject Matter of Software Copyright 159
2. Protection Requirements 165
a) Intellectual Creation 165
b) Originality (Individuality) 167
3. Term of Protection
VI. Conclusion 176

Chapter 5: Findings of the Interview Series 178
I. Software Development 178
A. The Relevance of Software Engineering 178
B. The Development Process 179
1. The Process 180
2. Duration 185
C. Programming in Particular 186
1. Selection of a Programming Language 186
2. Adaptability of Computer Programs 189
II. Software Commercialization 191
A. The Software Market 191
1. Offering on the International Market 191
2. Free Competition 193
B. Distribution Models 197
C. Excursus: The Significance of Open Source for Software Engineering 198
1. Chances and Benefits 200
2. Risks and Negative Effects 201
3. Closing Commentary 203
D. Product Life Cycle 204
III. Know-how in Software Engineering 206
A. The Significance of Knowledge and Know-how 206
B. Knowledge Transfer and Protection 210
IV. Creativity and Innovation in Software Engineering 212
A. Creativity 212
1. What is Creativity in Software Engineering? 212
2. The Impact of Toolboxes, Libraries and Assistance on Creativity 214
B. Innovation 215
1. What is Innovation? 216
2. Inventions in Software Engineering in Particular 218
3. Triviality as Contrary to Innovation 220
4. Discovering the Innovative Momentum 223
V. The Legal Protection of Computer Programs 224
A. The Significance of Software Protection 224
B. Experiences with the Current Legal Framework 228
1. Positive Aspects 228
2. Negative Experiences and Suggestions for Improvement 229
a) Computer Programs as Intellectual Properties 230
b) The Time Factor 230

c) Commissioned Work 232
d) Patentability of Computer Programs 233
aa) Machines vs. Computers 234
bb) Requirements for Patenting 235
cc) The Power to Block and Abuse 236
dd) Patenting as the Discipline of Kings 237
C. The Optimal Framework 238
1. The Function of Software Protection 239
2. Potential Objects of Protection 241
a) Valuable Components 241
b) The Suggested Objects for Software Protection 244
3. Starting Point for Legal Protection 247
4. The End of the Term of Protection 251
a) Points of Reference 251
b) Adequate Protection Period 253
5. Setting the Limits: Admissible and Inadmissible Third-Party Interventions 255
a) Delimiting Second-Hand Works: Where Do they Offer a Contribution? 255
b) Translations of the Source Code in Particular 258
c) Limiting Extensive Rights of Standard-Essential Patents and Fostering Incremental Improvements 259
aa) Standard-Essential Developments 259
bb) Incremental Improvements 260
d) Second-Hand Works: How To Compare Computer Programs 261
VI. Law Enforcement: Infringements and Legal Disputes 264
A. Infringements 264
B. Legal Disputes 265
VII. Summary 268

Chapter 6: Discussion of Selected Problems 276
I. Preface: Is there a Future for Legal Software Protection? 276
II. Copyright and Patent Protection: Hybrid Model or Copyright only? 282
III. The Protection Scope in Copyright and Patent Law 288
A. Circumscribing the Potential Subject Matter 289
1. Discovering Creativitiy and Inventiveness in Software Engineering 289
a) Creativity in Software Engineering 289
b) Inventiveness in Software Engineering 292

2. Protected Interests 293
a) Know-How and Resources 293
b) Know-How and Resource Protection in Copyright and Patent Law 296
B. The Protection Requirements in Copyright and Patent Law 296
C. Potential Subjects of Legal Software Protection 301
1. Whole Software Product 303
2. Code 304
3. Algorithm 306
4. (Graphical) User Interface 309
5. Look-and-Feel 312
6. Features and Functions 315
7. Development Documentation 316
8. Tabular Summary 319
D. Not Protected Elements in Particular 320
1. Distinguishing between Ideas and Expressions 320
2. Standards, Necessities and Best Practices 324
3. The Blackbox Test to Identify Unprotectable Elements 330
E. How to Integrate Newer Development Trends 335
1. Incremental Extension 337
2. Inner Change 341
F. Conclusion 343
IV. Term of Protection 344
A. Starting Point for Legal Protection 345
1. The Problem 346
2. Potential Starting Point for Linearly Developed Programs 347
3. Potential Starting Point for Non-Linear Development Approaches 350
a) Spiral Development 350
b) Inner Changes 352
B. Expiration 354
1. The Duration of the Protection Term 355
2. Connection Point 361
C. Conclusion 363
V. Excursus: Second-Hand and Dependent Creations 363
A. The Right Holder's Right to Protect a Work from Further Processing 365
1. Edited and Derivative Works 366
2. On Translations in Particular 368
B. The Community's Right to Profit from an Exclusive Property 371
1. Standard-Essential Developments 372

a) Types of Standards 374
b) The Antitrust Approach 375
c) The Patent Law Approach 381
aa) Based on a Superior Public Interest 382
bb) To Mitigate Negative Effects of Patenting 385
cc) Notes on the Possible Licensing Procedure and Terms 385
2. Incremental Improvements 388
a) Insufficient Regulation in Art. 31 lit. l TRIPS 390
b) Technical Advance 391
c) Important Advancement 394
d) Considerable Economic Significance 395
e) Notes on the Possible Licensing Procedure and Terms 398
VI. Conclusion 400

Chapter 7: Prospect and Closing 407
I. Scientific Contribution 407
II. Key Merits 408
III. Area for Action De Lege Lata and De Lege Ferenda 409
A. De Lege Lata 409
B. De Lege Ferenda 412
IV. Limitations and Perspective 415

List of Abbreviations

Only abbrevations which are used in this thesis are listed below.

ACM	Assosiation for Computing Machinery
AIPPI	Association International pour la Protection de la Propriété Industrielle
Art.	article
Austrian GMG	Austrian Gebrauchsmustergesetz of April 1, 1994, BGBl. 1994/211 (status as of June 14, 2018)
BAG	Bundesarbeitsgericht (German Federal Labour Court)
BB	Betriebs-Berater (periodical journal for law and economics)
Beck RS	Beck Rechtsprechung (collecton of jurisprudence)
BGE	Leitentscheide des Bundesgerichts (published ruling of the Swiss Federal Supreme Court)
BGer	Bundesgerichtsentscheid (unpublished ruling of the Swiss Federal Supreme Court)
BGBl	Bundesgesetzblatt (German federal law gazette)
BGH	Bundesgerichtshof (German Federal Court of Justice)
BGHZ	Entscheidungen des Bundesgerichtshofs in Zivilsachen (collection of the decisions of the civil law court division of the German Federal High Court of Justice)
BPatG	Bundespatentgericht (German Federal Patent Court)
BPatGer	Bundespatentgericht (Swiss Federal Patent Court)
BT-Drucks	Drucksachen und Plenarprotokolle des Bundestages (documents of the German Parliament)
c.	consideration
CD	compact disc
GC	General Court
CHF	Swiss Franc(s)
Cir.	Circuit Court in the United States
CJEU	Court of Juctice of the European Union

Computer Program Directive	Council Directive (91/250/EEC) of 14 May 1991 on the legal protection of computer programs, implemented 1 January 1993, in force until 23 April 2009, substituted by the Directive of the European Parliament and of the Council (2009/24/EC) of 23 April 2009
Cong.	U.S. Congress
Copyright Directive	Directive of the European Parliament and of the Council (2001/29/EC) of 22 May 2001, on the harmonisation of certain aspects of copyright and related rights in the information society
DIN	Deutsches Institut für Normierung (German Institute for Standardization)
ECDR	European Copyright and Design Reports
EC Copyright Term of Protection Directive	Council Directive 93/98/EEC of 29 October 1993, harmonizing the term of protection of copyright and certain related rights
ECJ	European Court of Justice
E-Commerce Directive	Directive of the European Parliament and of the Council (2000/31/EC) of 8 June 2000 on certain legal aspects of information society services, in particular electronic commerce, in the Internal Market
ed.	editor
eds.	editors
EPC	European Patent Convention of 5 October 1973, revised 29 November 2000 (status as of June, 2016)
EPO	European Patent Office
EPO T	decisions of the European Patent Office and its Board of Appeal
EUR	Euro(s)
EWCA	Court of Appeal of England and Wales
EWHC	High Court of England and Wales
f.	following page
Fed. Cir.	U.S. Court of Appeals for the Federal Circuit
ff.	following pages
fn.	footnote

French IP Code	French Intellectual Property Code of 1 July 1992, no 92–597 (status as of August 1, 2019)
German DesG	German Design Act of 24 February 2014, BGBl. I S. 122 (status as of July 17, 2017)
German GebrMG	German Gebrauchsmustergesetz of 28 August 1986, BGBl. I 1455 (status as of July 17, 2017)
German GeschGehG	German Trade Secret Protection Act of 18 April 2019, BGBl. I S. 466
German MarkenG	German Markengesetz of 25 October 1994, BGBl. I S. 3082 (status as of December 11, 2018)
German PatG	German Patent Code of 5 May 1938, revised 16 December 1980, BGBl. 1981 I 1 (status as of October 8, 2017)
German UrhG	German Copyright Act of 9 September 1965, BGBl. I S. 1273 (status as of November 28, 2018)
German UWG	German Act against Unfair Competition of 3 July 2004, BGBl. I S. 3714 (status as of April 18, 2019)
GRUR	Gewerblicher Rechtsschutz und Urheberrecht (periodical)
HGer	Handelsgericht (Commercial Court of cantons in Switzerland)
ibid.	ibidem
ICT	information and communication technology
icw	in conjunction with
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	intellectual property
ISO	International Organization for Standardization
Lanham Act	Lanham Trademark Act of 5 July 1946, 60 Stat. 427, codified at 15 U.S.C. § 1051 ff.
LG	Landgericht (lower regional courts in Germany)
lit.	litera
mio.	million(s)
MMR	Multimedia und Recht (periodical)
N	marginal note
NJW	Neue Juristische Wochenschrift (periodical)
no.	number

OECD	Organisation for Economic Co-operation and Development
OGer	Obergericht (High Court of cantons in Switzerland)
OLG	Oberlandsgericht (higher regional courts in Germany)
openJUR	legal database for German decisions
OR	Swiss Code of Obligations, Federal Act on the Amendment of the Swiss Civil Code of 30 March 1911, SR 220 (status as of April 1, 2017)
para.	paragraph
Paris Convention	Revised Paris Convention for the Protection of Industrial Property of 14 July 1967 (status as of March 20, 1983)
PCT	Patent Cooperation Treaty of 19 June 1970, revised 3 October 2001 (status as of June 28, 2019)
PMMBl	Patent-, Muster, und Markenblatt (publication of patent applications and granted patents in Switzerland before 1 July 2008)
RBC	Revised Berne Convention for the Protection of Literary and Artistic Works of 9 September 1886, revised in Paris on July 24, 1971 (status as of June 27, 2019)
RPC	Reports of Patent, Design and Trade Mark Cases
S.	U.S. Senate
sect.	section
Ses.	Session of the U.S. Congress or U.S. Senate
sic!	Swiss Law Review for Intellectual Property, Information and Competition Law
SIWR	Schweizerisches Immaterialgüter- und Wettbewerbsrecht (book series)
SR	Systematische Rechtsammlung (classified compilation in Switzerland)
Swiss CopA	Swiss Federal Act on Copyright and Related Rights of 9 October 1992, SR 231.1 (status as of 1 January, 2020)
Swiss DesG	Swiss Designs Act of 5 October 2001, SR 232.2 (status as of January 1, 2017)
Swiss DesR	Swiss Designs Regulation of 8 March 2001, SR 232.12 (status as of January 1, 2017)
Swiss MSchG	Swiss Trade Mark Protection Act of 28 August 1992, SR 232.11 (status as of April 1, 2019)

Swiss PatG	Swiss Patents Act of 25 June 1954, SR 232.14 (status as of January 1, 2012)
Swiss UWG	Swiss Act against Unfair Competition of 19 December 1986 SR 241.00 (status as of July 1, 2016)
TRIPS Agreement	Agreement on Trade-Related Aspects of Intellectual Property Rights of 15 April 1994, Annex 1C to the Agreement establishing the World Trade Organization (status as of January 23, 2017)
UCC	Universal Copyright Convention of 24 July 1971 (status as of April 23, 2010)
UFITA	Institut für Urheber- und Medienrecht (series of publication), previously named Archiv für Urheber-, Film-, Funk- und Theaterrecht
UK IP Act	United Kingdom Copyright, Designs and Patents Act of 15 November 1988 (status as of October 11, 2018)
UKSC	United Kingdom Supreme Court
U.S. Code	Code of Laws of the United States of America of 30 June 1926 (status as of 2018)
U.S. Constitution	Constitution of the United States of 4 March 1789 (status as of April 18, 1999)
USD	U.S. Dollar(s)
UTSA	Uniform Trade Secrets Act of 1979, amended 2 August 1985
vol.	volume
WCT	WIPO Copyright Treaty of 20 December 1996 (status as of June 27, 2019)
WIPO	World Intellectual Property Organization
WTO	World Trade Organization

References

Bibliography

Allison John R./Lemley Mark A./Schwartz David L., Our divided patent system, the University of Chicago Law Review, vol. 82 no. 3, 2015, pp. 1073 ff.

Andrews Philip Walter Sawford, Manufacturing Business, London 1949.

Apel Sven, The Role of Features and Aspects in Software Development, Dissertation at the Otto-von-Guericke University, Magdeburg 2007.

Armstrong John A., Trends in Global Science and Technology and What They Mean for Intellectual Property Systems, in: Wallerstein M./Mogee M./Schoen R. (eds.), Global Dimensions of Intellectual Property Rights in Science and Technology, Washington (DC) 1993, pp. 192 ff.

Arrow Kenneth J., Economic Welfare and the Allocation of Resources for Invention, Readings in industrial economics, London 1972, pp. 219 ff.

Associaftion Internationale pour la Protection de la Propriété Industrielle (AIPPI), Protection of Computer-Software, yearbook III of 1975, question 57 of the 29th Congress of San Francisco (May 3-7, 1975), Geneva 1975, found online on: <https://www.aippi.fr/upload/Q1%20-%2089%20/rs57english.pdf> (retrieved September 6, 2021), cited as Aippi.

Atteslander Peter, Methoden der empirischen Sozialforschung, 13th edition, Berlin 2010.

Banasevic Nicholas, The Implications of the Court of Justice's Huawei/ZTE Judgment, Journal of European Competition Law & Practice, vol. 6 no. 7, 2015, pp. 463 ff.

Barrelet Denis/Egloff Willi, Das neue Urheberrecht. Kommentar zum Bundesgesetz über das Urheberrecht und verwandte Schutzrechte, 3rd edition, Berne 2008.

Barry Evelyn J./Kemerer Chris F./Slaughter Sandra A., On the Uniformity of Software Evolution Patterns, Proceedings on the 25th International Conference on Software Engineering, Electrical and Electronics Engineers (IEEE) Computer Society Press, New York 2003, pp. 106 ff., found online at <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1201192> (retrieved September 6, 2021).

Bauknecht Kurt/Zehnder Carl August, Grundzüge der Datenverarbeitung. Methoden und Konzepte für die Anwendungen. Leitfäden der angewandten Informatik, Stuttgart 1989.

Becker Mario/Haberfellner Reinhard/Liebetrau Georg/Voessner Siegfried: EDV-Wissen für Anwender. Das Informatik-Handbuch für die Praxis, 13th edition, Zurich/Stuttgart 2004, cited as Becker et al.

Beier Friedrich-Karl, Die Bedeutung des Patentsystems für den technischen, wirtschaftlichen und sozialen Fortschritt, GRUR International, 1979, pp. 227 ff.

Belady Laszlo A./Lehman Meir M., A Model of Large Program Development, IBM Systems Journal, vol. 15 no. 3, 1976 , pp. 225 ff.

Bell Abraham/Parchomovsky Gideon, Reinventing Copyright and Patent, Michigan Law Review, vol. 113, 2014, pp 231 ff.

Bell Thomas E./Thayer Thomas A., Software Requirements: Are They Really a Problem?, Proceedings of the 2nd international conference on Software engineering, IEEE Computer Society Press, New York 1976, pp. 61 ff.

Benkard Georg/Ehlers Jochen/Kinkeldey Ursula (eds.), Europäisches Patentübereinkommen, Munich 2012, cited as Commentary to the EPC (editor).

Benoit Kenneth/Wiesehomeier Nina, Expert Judgments, in: Pickel S./Pickel G./Lauth H.-J./Jahn D. (eds.), Methoden der vergleichende Politik- und Sozialwissenschaft. Neue Entwicklungen und Anwendungen, Wiesbaden 2009, pp. 497 ff.

Beresford Keith, Patenting Software under the European Patent Convention, London 2000.

Bock Richard, Simultane Produktentwicklung – Konzepte und Realisierungsalternativen, in: Scheer A.-W., Handbuch Informationsmanagement, Wiesbaden 1993, pp. 221 ff.

Bodewig Theo, Einige Überlegungen zur Erschöpfung bei Zwangslizenzen an standardessentiellen Patenten, GRUR International, 2015, pp. 626 ff.

Boecker Lina B., Computerprogramme zwischen Werk und Erfindung. Eine wettbewerbsorientierte Analyse des immaterialgüterrechtlichen Schutzes von Computerprogrammen unter besonderer Berücksichtigung von Open Source-Software, Dissertation at the University of Berlin, in: Mestmaecker E.-J./Moeschel W./Hellwig M., Wirtschaftsrecht und Wirtschaftspolitik, Band 229, Baden-Baden 2009.

Boehm Barry W., Software Engineering Economics, Englewood Cliffs River (NJ) 1981, cited as Boehm (1981).

Boehm Barry W., A Spiral Model of Software Development and Enhancement, Computer, IEEE Computer Society Press, vol. 21 no. 5, 1988, pp. 61 ff., cited as Boehm (1988).

Bogner Alexander/Menz Wolfgang, Das theoriengenerierende Experteninterview – Erkenntnisse, Wissensformen, Interaktion, in: Bogner A./Littig B./Menz W. (eds.), Experteninterviews. Theorien, Methoden, Anwendungsfelder, 3rd edition, Wiesbaden 2009, pp. 61 ff., cited as Bogner/Menz (2009a).

Bogner Alexander/Menz Wolfgang, Experteninterview und der Wandel der Wissensproduktion, in: Bogner A./Littig B./Menz W. (eds.), Experteninterviews. Theorien, Methoden, Anwendungsfelder, 3rd edition, Wiesbaden 2009, pp. 7 ff., cited as Bogner/Menz (2009b).

Bohnsack Ralf/Geimer Alexander/Meuser Michael, Hauptbegriffe qualitativer Sozialforschung, 4th edition, Opladen/Toronto 2018.

Bornhauser Jonas, Anwendungsbereich und Beschränkung des urheberrechtlichen Vervielfältigungsrechts im digitalen Kontext, Dissertation at the University of Zurich, Berne 2010.

Bowen Glenn A., Document Analysis as a Qualitative Method, Qualitative Research Journal, vol. 9 no. 2, 2009, pp. 27 ff.

Brandi-Dohrn Matthias, Zur Reichweite und Durchsetzung des urheberrechtlichen Softwareschutzes, GRUR, 1985, pp. 179 ff.

Braumoeller Bear F./Goertz Gary, The Methodology of Necessary Conditions, American Journal of Political Science, vol. 44 no. 4, 2000, pp. 844 ff.

Briner Alfred, Patentierungsvoraussetzungen, in: David L./von Bueren R. (eds.), SIWR IV, Basel/Geneva/Munich 2006, pp. 47 ff.

Buehler Lukas, Schweizerisches und internationales Urheberrecht im Internet, Dissertation of the University of Fribourg, Fribourg 1999.

Bullinger Hans-Joerg/Faehnrich Klaus-Peter/Ilg Rolf, Benutzungsoberflächen und Entwicklungswerkzeuge, in: Scheer A.-W., Handbuch Informationsmanagement, Wiesbaden 1993, pp. 939 ff.

Burkert Herbert/Hettich Peter/Thouvenin Florent, Eine kritische Geschichte des Informationsrechts – Erlebte, bevorstehende und versäumte Paradigmenwechsel im Recht zufolge Digitalisierung, in: Gschwend L./Hettich P./Mueller-Chen M./Schindler/Wildhaber I., Recht im digitalen Zeitalter. Festgabe Schweizerischer Juristentag 2015 in St. Gallen, Zurich/St. Gallen 2005, pp. 49 ff.

Buxmann Peter/Hess Thomas/Lehmann Sonja, Software as a Service, Wirtschaftsinformatik, vol. 50 no. 6, 2008, pp. 500 ff.

Calame Thierry, Grundlagen; Die Berechtigung an der Erfindung; Die Wirkung des Patents; Besonderheiten von computerimplmentierten Erfindungen, in: David L./von Bueren R. (eds.), SIWR IV, Basel/Geneva/Munich 2006, pp. 3 ff., 171 ff., 401 ff. and 651 ff., cited as Calame (2006).

Calame Thierry, Softwareschutz: Möglichkeiten und Grenzen, in: Internet-Recht und Electronic Commerce Law, 9. Tagungsband, Berne 2007, pp. 325 ff., cited as Calame (2007).

Carleton Dennis M., A Behaviour-Based Model for Determining Software Copyright Infringement, Berkeley Technology Law Journal, vol. 10 no. 2, 1995, 405 ff.

Charmaz Kathy, Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis, London 2013.

Chen Lianping, Continuous Delivery: Huge Benefits, but Challenges Too, IEEE, vol. 32 no. 2, 2015, pp. 50 ff.

Cherpillod Ivan, L'objet du droit d'auteur. Etude critique de la distinction entre forme et idée, Dissertation at the University of Lausanne, Lausanne 1985, cited as Cherpillod (1985).

Cherpillod Ivan, Urheberrecht: Geltungsbereich, in: David L./von Bueren R. (eds.), SIWR II/1, Basel/Geneva/Munich 2014, pp. 14 ff., cited as Cherpillod (2014).

Cicourel Aaron V., Method and Measurement in Sociology, London 1964.

Cockburn Alistair, Using Both Incremental and Iterative Development, STSC CrossTalk, USAF Software Technology Support Center, vol 21 no. 5, Utah 2008, pp. 27 ff.

Cohen Morris A./Eliasberg Jehoshua/Ho Teck-Hua, New Product Development: The Performance and Time-to-Market Tradeoff, Management Science, vol. 42 no. 2, 1996, pp. 173 ff.

Cohen Julie E./Lemley Mark A., Patent Scope and Innovation in the Software Industry California Law Review, vol. 89, 2007, pp. 1 ff.

Corbin Juliet/Strauss Anselm, Basics of Qualitative Research, 4th edition, Los Angeles 2015.

Cowan Robin/Jonard Nicolas, The Dynamics of Collective Invention, Journal of Economic Behavior & Organization, vol. 52 no. 4, 2003, pp. 513 ff.

Cross Nigel, Design Cognition: Results from Protocol and other Empirical Studies of Design Activity, in: Eastman C./McCracken M./Newstetter W. (eds.), Design Knowing and Learning: Cognition in Design Education, Atlanta (GA) 2001, pp. 79 ff.

David Paul A., Intellectual Property Institutions and the Panda's Thumb: Patent, Copyright, and Trade Secrets in Economic Theory and History, in: Wallerstein M./Mogee M./Schoen R. (eds.), Global Dimensions of Intellectual Property Rights in Science and Technology, Washington (DC) 1993, pp. 29 ff.

Davidson Ducan M., The Future of Software Protection: Common Law, Uncommon Software, University of Pittsburgh Law Review, vol. 47, 1985, pp. 1037 ff.

Deeke Axel, Experteninterviews – ein methodologisches und forschungspraktisches Problem. Einleitende Bemerkungen und Fragen zum Workshop, in: Brinkmann Ch./Deeke A./Voelkel B. (eds.), Experteninterviews in der Arbeitsmarktforschung, Nuernberg 1995, pp. 7 ff.

Denzin Norman K., The Art and Politics of Interpretation, in: Denzin Norman K./Lincoln Yvonna S., Collecting and Interpreting Qualitative Materials, Thousand Oaks 1998, pp. 313 ff.

Dessemontet François, Einführung: Immaterialgüterrecht und Privatrecht; Schutzdauer, in: David L./von Bueren R. (eds.), SIWR I/1, Basel/Geneva/Munich 1995, pp. 1 ff. and 315 ff.

Denzin Norman K./Lincoln Yvonna S., The Sage Handbook of Qualitative Research, 5th edition, Los Angeles 2005.

Dijkstra Edsger W., A Discipline of Programming, Englewood Cliffs (NJ) 1976.

Dorr Robert C./Munch Christopher H., Protecting Trade Secrets, Patents, Copyrights, and Trademarks, New York (NY) 1995.

Dreier Thomas, Der Urheberrechtsschutz für Computerprogramme im Ausland – Rechtsfragen und Tendenzen in Rechtsprechung und Gesetzgebung, GRUR Int., 1988, pp. 476 ff., cited as Dreier (1988).

Dreier Thomas, Verletzung urheberrechtlich geschützter Software nach der Umsetzung der EG-Richtlinie, GRUR, 1993, pp. 781 ff., cited as Dreier (1993).

Dubin Joseph S., The Universal Copyright Convention, California Law Review, vol. 42 no. 1, 1954, pp. 89 ff.

Dutfield Graham/Suthersanen Uma, Global Intellectual Property Law, Cheltenham (UK)/Northhampton (MA) 2008.

Ebbinghaus Bernhard, Mehr oder weniger? Quantitativer versus Qualitativer Vergleich, in: Pickel S./Pickel G./Lauth H.-J./Jahn D. (eds.), Methoden der vergleichende Politik- und Sozialwissenschaft. Neue Entwicklungen und Anwendungen, Wiesbaden 2009, pp. 197 ff.

Edvardsson Bo, Quality in New Service Development: Key Concepts and a Frame of Reference, International Journal of Production Economics, no. 52/1-2, Oxford 1997, pp. 31 ff.

Ensthaler Juergen/Moellenkamp Heinz T., GRUR, 1994, pp. 151 ff.

Erlandson David A./Harris Edward L./Skipper Barbara L./Allen Steve D., Doing Naturalistic Inquiry: A Guide to Methods, Newbury Park/London/New Delhi 1993.

Ernst Stefan, Die Verfügbarkeit des Source Codes – Rechtlicher Know-how-Schutz bei Software und Webdesign, MMR, 2001, pp. 208 ff.

Errat Judy A./Gowling, Strathy & Henderson, A Study on the Patent Law Standard of Non-obviousness, Ottawa 1996.

Ewusi-Mensah Kweku, Critical Issues in Abandoned Information Systems Development Projects, Communications of the ACM, vol. 40 no. 9, 1997, pp. 74 ff.

Finch John H., The Role of Grounded Theory in Developing Economic Theory, Journal of Economic Methodology, vol. 9 no. 2, 2002, pp. 213 ff.

Fisher Francis D., The Electronic Lumberyard Builders' Rights: Technology, Copyrights, Patents, and Academe, Change: The Magazine of Higher Learning, vol. 21 no. 3, 1989, pp. 12 ff.

Floyd Robert W., The Paradigms of Programming, Communications of the ACM, vol. 22 no. 8, 1979, pp. 455 ff.

Foegen Malte/Meyser Astrid/Gansser Caroline/Croome David/Raak Claudia/Battenfeld Joerg/Kroell Anna K./Fritsch-Leopoldt Christoph/Porro Simon, Der ultimative Scrum Guide, Darmstadt 2014, cited as Foegen et al.

Forstmoser Peter, Vertragsprobleme im Bereich der EDV, Sysdata und Bürotechnik, vol. 8-9, Liestal 1975, pp. VI ff.

Frei Alexandra, Softwareschutz durch Patentrecht, in: Thomann F. H./Rauber G. (eds.), Softwareschutz, Bern 1998, pp. 97 ff.

Froehlich-Bleuler Gianni, Indirekte Nutzung von Computerprogrammen, sui-generis, Zurich 2018, pp. 464 ff.

Fuegi John/Francis Jo, Lovelace & Babbage and the Creation of the 1843 'Notes', IEEE Computer Society Press, vol. 25 no. 4, 2003, pp. 16 ff., found online at <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1253887> (retrieved September 6, 2021).

Fusco Stefania, Is In Re Bilski a Déjà Vu?, Stanford Technology Law Review, Palo Alto 2009, pp. 143 ff.

Galal Galal Hassan, From Contexts to Constructs: The Use of Grounded Theory in Operationalising Contingent Process Models, European Journal of Information Systems 10 no. 1, Abingdon 2001, pp. 2 ff.

George Alexander L./Bennett Andrew, Case Studies and Theory Development in the Social Sciences, Cambridge (MA) 2005.

Giddens Anthony, Leben in einer post-traditionalen Gesellschaft, Soziale Welt, vol. 44 no. 4, 1993, pp. 445 ff.

Girtler Roland, Methoden der Feldforschung, 4th edition, Vienna/Cologne/Weimar 2001.

Glaser Barney G./ trauss Anselm L., The Discovery of Grounded Theory. Strategies for Qualitative Research, London/New York 2017.

Gmehlich Rainer/Rust Heinrich, Mehr als nur Programmieren. Eine Einführung in die Informatik, Braunschweig/Wiesbaden 1993.

Goldstein Kenneth, Getting in the Door: Sampling and Completing Elite Interviews, Political Science and Politics, vol. 35 no. 4, 2002, pp. 669 ff.

Gordon Raymond L., Interviewing: Strategy, Techniques and Tactics, Homewood (IL) 1969.

Govoni Carlo, Der urheberrechtliche Schutz von Computerprogrammen, Aktuelle Juristische Praxis, 1993, pp. 569 ff.

Grossenbacher Roland, Die Entwicklung des Welturheberrechtsabkommens im Hinblick auf den Beitritt der Sowjetunion, Dissertation at the University of Zurich, Zurich 1977.

Gruetzmacher Malte, Gebrauchtsoftware und Übertragbarkeit von Lizenzen. Zu den Rechtsfragen auch jenseits der Erschöpfungslehre, Computer und Recht, vol. 23 no. 9, 2007, pp. 549 ff.

Haberstumpf Helmut, Computerprogramm und Algorithmus. Zur Vereinbarkeit des urheberrechtlichen Schutzes für Computerprogramme mit den Erfordernissen des wissenschaftlichen Fortschritts, UFITA, vol. 95, 1983, pp. 221 ff., cited as Haberstumpf (1983).

Haberstumpf Helmut, Der urheberrechtliche Schutz von Computerprogrammen, in: Lehmann M. (ed.), Rechtsschutz und Verwertung von Computerprogrammen, 2nd edition, Cologne 1993, pp. 69 ff., cited as Haberstumpf (1993).

Hage Jerald, Techniques and Problems of Theory Construction in Sociology, New York 1972.

Harison Elad, Intellectual Property Rights Innovation and Software Technologies. The Economics of Monopoly Rights and Knowledge Disclosure, Cheltenham 2008.

Hart Chris, Doing a Literature Review: Releasing the Social Science Research Imagination, 2nd edition, London/Thousand Oaks/New Delhi/Singapore 2018.

Heinemann Andreas, Immaterialgüterschutz in der Wettbewerbsordnung: Eine grundlagenorientierte Untersuchung zum Kartellrecht des geistigen Eigentums, Jus Privatum, vol. 65, Tuebingen 2002, cited as Heinemann (2002).

Heinemann Andreas, Compulsory Licences and Product Integration in European Competition Law – Assessment of the European Commission's Microsoft Decision, International Review of Intellectual Property and Competition Law, vol. 36, 2005, pp. 63 ff., cited as Heinemann (2005).

Heinemann Andreas, Gefährdung von Rechten des geistigen Eigentums durch Kartellrecht? Der Fall „Microsoft" und die Rechtsprechung des EuGH, GRUR, 2006, pp. 705 ff., cited as Heinemann (2006).

Heinemann Andreas, Wettbewerb auf den Märkten der Informationstechnologie – Die Perspektive des Europäischen Kartellrechts, in: Epiney A./Diezig St. (eds.), Schweizerisches Jahrbuch für Europarecht 2012/2013, Zurich/Basel/Geneva 2013, pp. 355 – 376, cited as Heinemann (2013).

Heinemann Andreas, Standardessenzielle Patente in Normenorganisationen. Kartellrechtliche Vorgaben für die Einlösung von Lizenzierungsversprechen, GRUR, 2015, pp. 855 ff., cited as Heinemann (2015).

Heinrich Peter, DesG/HMA: Kommentar zum schweizerischen Designgesetz, den entsprechenden Bestimmungen des Haager Musterschutzabkommens und weiteren Erlassen, 2nd edition, Zurich 2014, cited as Orell Fuessli Commentary to the Swiss DesG.

Heinrich Peter, PatG/EPÜ: Kommentar zum Schweizerischen Patentgesetz und den entsprechenden Bestimmungen des Europäischen Patentübereinkommens synoptisch dargestellt, 3nd edition, Berne 2018, cited as Staempfli Commentary to the Swiss PatG/EPC.

Helfferich Cornelia, Die Qualität qualitativer Daten. Manual für die Durchführung qualitativer Interviews, 3rd edition, Wiesbaden 2011.

Hepp Dieter/Mueller Christoph/Herrmann Torbjoern, Softwareschutz und Softwareverträge in einzelnen Ländern: Schweiz, in: Ullrich H./ Lejeune M. (eds.), Der internationale Softwarevertrag nach deutschem und ausländischem Recht, 2nd edition, Frankfurt am Main 2006, pp. 1151 ff.

Hess Thomas/Wilde Thomas, Forschungsmethoden der Wirtschaftsinformatik. Eine empirische Untersuchung, Wirtschaftsinformatik, vol. 49 no. 4, 2007, 280 ff.

Hilti Christian/Pedrazzini Mario M., Europäisches und Schweizerisches Patent- und Prozessrecht, 3rd edition, Bern 2008.

Hilty Reto M., Der Schutz von Computerprogrammen – nationale und internationale Normen auf dem Prüfstand des Internets, sic!, 1997, pp. 128 ff., cited as Hilty (1997).

Hilty Reto M., Urheberrecht, Bern 2011, cited as Hilty (2010).

Hilty Reto M., Softwareurheberrecht statt Softwarepatente? Forderungen der deutschen Politik unter der Lupe, in: Alexander Ch./Bornkamm J./Buchner B./Fritzche J./Lettl T., Festschrift für Helmut Köhler zum 70. Geburtstag, Munich 2014, pp. 289 ff., cited as Hilty (2014).

Hilty Reto M., Innovationsförderung durch Schutzbegrenzungen – ein Plädoyer für die Zwangslizenz, in: Grolimund P./Koller A./Loacker L. D./Portmann W., Fest Festschrift für Anton K. Schnyder zum 65. Geburtstag, Zurich 2018, pp. 1179 ff., cited as Hilty (2018).

Hilty Reto M./Geiger Christophe, Patenting Software? A Judicial and Socio-Economic Analysis, International Journal of Intellectual Property Management, Olney 2005, pp. 615 ff., cited as Hilty/Geiger (2005).

Hilty Reto M./Geiger Christophe, Towards a New Instrument of Protection for Software in the EU? Learning the Lessons from the Harmonization Failure of Software Patentability, in: Ghidini G./Arezzo E. (eds.), Biotechnology aAnd Software Patent Law – A Comparative Review of New Developments, Cheltenham UK/Northhampton (MA) 2011, pp. 153 ff., cited as Hilty/Geiger (2011).

Hilty Reto M./Koeklue Kaya, Reichweite des Rechtsschutzes von Computerprogrammen – Eine Kritik an der EuGH-Rechtsprechung, in: Buescher W./Erdmann W./Haedicke M./Koehler H. Loschelder M. (eds.), Festschrift für Joachim Bornkamm zum 65. Geburtstag, Munich 2014, pp. 797 ff.

Hilty Reto M./Slowinski Peter R., Standardessentielle Patente – Perspektiven ausserhalb des Kartellrechts, GRUR International, 2015, pp. 781 ff.

Hoepfl Marie C., Choosing Qualitative Research: A Primer for Technology Education Researchers, Journal of Technology Education, vol. 9 no. 1, 1997, pp. 47 ff.

Hoeren Thomas, Internetrecht, 3rd edition, Cologne 2018.

Hommel Guenter/Jaehnichen Stefan/Koster Cornelis H. A., Methodisches Programmieren, Berlin/New York 1983, cited as Hommel et al.

Hopf Christel, Die Pseudo-Exploration – Überlegungen zur Technik qualitativer Interviews in der Sozialforschung, Zeitschrift für Sozialforschung, vol. 7 no. 2, 1978, pp. 97 ff.

Horns Axel, Anmerkungen zu begrifflichen Fragen des Softwareschutzes, GRUR, 2001, pp. 1 ff.

Humble Jez/Farley David, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Boston 2011.

Jaenich Michael, Sonderrechtsschutz für geschäftliche Methoden, GRUR, 2003, pp. 483 ff.

Jersch Ralf, Ergänzender Leistungsschutz und Computersoftware: Rechtsschutz für innovative Arbeitsergebnisse durch UWG und BGB, Dissertation at the Westfaelischen Wilhelms-University of Muenster, Schriftenreihe Recht und Computerpraxis, vol. 3, Munich 1993.

Kooths Stefan/Langenfurth Markus/Kalway Nadine, Open-SourceSoftware. Eine volkswirtschaftliche Bewertung, MICE Economic Research Studies, vol. 4, 2003, cited as Kooths et al.

Katz Michael L./Shapiro Carl, Systems Competition and Network Effects, Journal of Economic Perspectives, no, 2, Nashville 1994, pp. 93 ff.

Kawulich Barbara B., Participant Observation as a Data Collection Method, Forum Qualitative Social Research, vol. 6 no. 2, 2005, article 43.

Kindermann Manfred, Vertrieb und Nutzung von Computersoftware aus urheberrechtlicher Sicht, GRUR, 1983, pp. 150 ff.

Kitch Edmund W., The Nature and Function of the Patent System, The Journal of Law & Economics, vol. 20 no. 2, 1977, pp. 265 ff.

Klemperer Paul, How Broad Should the Scope of Patent Protection Be?, The RAND Journal of Economics, vol. 21 no 1, 1990, pp. 113 ff.

Koehler Reimar, Der urheberrechtliche Schutz der Rechenprogramme, Urheberrechtliche Abhandlungen, vol. 8, Munich 1968.

Koerber Torsten, Abuse of a Dominant Position by Legal Actions of Owners of Standard-Essential Patents: Huawei Technologies Co. Ltd v. ZTE Corp., Common Market Law Review, vol. 53 no. 4, 2016, pp. 1107 ff.

Koglin Olaf, Opensourcerecht. Die urheber- und schuldrechtlichen Beziehungen zwischen Lizenzgeber und Lizenznehmer bei Open Source Software am Beispiel der General Public License (GPL), in: Costede J./Spindler G., Schriften zum Wirtschafts- und Medienrecht, Steuerrecht und Zivilprozessrecht, vol. 31, Franfurt am Main 2007.

Koreimann Dieter S., Grundlagen der Software-Entwicklung, 3rd edition, Munich/Vienna/Oldenbourg 2000.

Krasser Rudolf, Erweiterung des patentrechtlichen Erfindungsbegriffs?, GRUR, 2001, pp. 959 ff.

Kribber Klaus-Dieter, Berücksichtigung gewerblicher Schutzrechte in Softwareentwicklungsprojekten, Fachberichte des Fachbereichs Elektrotechnik, vol. 1, 1998, found online on <https://www.yumpu.com/de/document/read/5858593/berucksichtigung-gewerblicher-schutzrechte-in-software-> (retrieved September 6, 2021).

Krueger Charles W., Software Reuse, ACM Computing Surveys, vol. 24 no. 2, 1992, pp. 131 ff.

Krueger Wilhelm/Pfeiffer Peter, Eine konzeptionelle und empirische Analyse der Informationsstrategien und der Aufgaben des Informationsmanagements, Zeitschrift für betriebswirtschaftliche Forschung, vol. 43, 1991, pp. 21 ff.

Kummer Max, Das urheberrechtlich schützbare Werk, Berne 1968.

Kur Annette, Auswirkungen des neuen Geschmacksmusterrecht auf die Praxis, GRUR, 2002, pp. 661 ff., cited as Kur (2002).

Kur Annette, Protection of Graphical User Interfaces Under European Design Legislation, International Review of Intellectual Property and Competition Law, vol. 34 no. 1, 2003, pp. 50 ff. cited as Kur (2003).

Landes William M./Posner Richard A., An Economic Analysis of Copyright Law, The Journal of Legal Studies, vol. 18 no. 2, 1989, pp. 325 ff., cited as Landes/Posner (1989).

Landes William M./Posner Richard A., The Economic Structure of Intellectual Property Law, Cambridge (MA) 2003, cited as Landes/Posner (2003).

Lang Johannes, Patent Protection for E-Commerce Methods in Europe, Computer and Telecommunications Law Review, vol. 6 no. 5, 2000, pp. 117 ff.

Langley Ann, Strategies for Theorizing From Process Data, The Academy of Management Review, vol. 24 no. 4, 1999, pp. 691 ff.

Larman Craig/Basili Victor R., Iterative and Incremental Development: A Brief History, IEEE Computer Society Press, vol. 36 no. 6, 2003, pp. 47 ff.

Laurie Ronald S., Patentability of Computer Programs in the USA, in: Meijboom A., The Law of Information Technology in Europe: A Comparison with the USA, Deventer 1991.

Lehmann Michael, Der Rechtsschutz von Computerprogrammen in Deutschland, NJW, 1988, pp. 2419 ff., cited as Lehmann (1988).

Lemley Mark A., Convergence in the Law of Software Copyright, Berkeley Technology Law Review, vol. 10 no. 1, 1995, pp. 1 ff., cited as Lemley (1995).

Lemley Mark A., Intellectual Property Rights and Standard-Setting Organizations, California Law Review, vol. 90 no. 6, pp. 1889 ff., cited as Lemley (2002).

Lemley Mark A., Software Patents and the Return of Functional Claiming, Wisconsin Law Review, 2013, pp. 905 ff., cited as Lemley (2013).

Lemley Mark A./Burk Dan L., Designing optimal software patents, in: Hahn R. (ed.), Intellectual Property Rights in Frontier Industries: Software and Biotechnology, Washington (DC) 2005, pp. 81 ff.

Lemley Mark A./Shapiro Carl, A Simple Approach to Setting Reasonable Royalties for Standards-Essential Patents, Berkeley Technology Law Journal, vol. 28, 2013, pp. 1135 ff.

Lemley Mark A./Menell Peter S./Merges Robert P./Samuelson Pamela/Carver Brian W., Software & Internet Law, 4th edition, New York 2011, cited as Lemley et al.

Lessig Lawrence, The Limits in Open Code: Regulatory Standards and the Future of the Net, Berkeley Technology Law Journal, vol. 14, 1999, pp. 759 ff.

Liebold Renate/Trinczek Rainer, Experteninterview, in: Kuehl St./Strodtholz P./Taffertshofer A. (eds.), Handbuch Methoden der Organisationsforschung. Quantitative und Qualitative Methoden, Wiesbaden 2009, pp. 32 ff.

Lincoln Yvonna S./Guba Egon G., Naturalistic Inquiry, Thousand Oaks 1985.

Littig Beate, Interviews mit Eliten – Interviews mit ExpertInnen: Gibt es Unterschiede?, in: Bogner A./Littig B./Menz W. (eds.), Experteninterviews. Theorien, Methoden, Anwendungsfelder, 3rd edition, Wiesbaden 2009, pp. 61 ff.

Lueck Gert, Rechtsschutz und Vertragsgestaltung bei Computer-Software aus Schweizer Sicht, Schriften zum Medien- und Immaterialgüterrecht, vol. 1, 1986, pp. 17 ff.

Lutz Martin Johannes, Der Schutz der Computerprogramme in der Schweiz, GRUR International, 1993, pp. 653 ff.

Machlup Fritz, Die wirtschaftlichen Grundlagen des Patentrechts, GRUR Ausland, 1961, pp. 373 ff.

Mamane David, Das kartellrechtliche Damoklesschwert über dem Immaterialgüterrecht, in: Olano O. (ed.), Liber amicorum für Felix. H. Thomann zum 80. Geburtstag, Zurich 2016, pp. 27 ff.

Marbach Eugen/Ducrey Patrik/Wild Gregor, Immaterialgüter- und Wettbewerbsrecht, 4th Edition, Berne 2017.

Marly Jochen, Praxishandbuch Softwarerecht: Rechtsschutz und Vertragsgestaltung, 7th edition, Munich 2018.

Marshall Catherine/Rossmann Gretchen B., Designing Qualitative Research, 6th edition, Los Angeles 2016.

Mathematical Programming Society, Report of the Committee on Algorithms and the Law, Optima, vol. 33 no. 2, 1991, pp. 1 ff.

Maume Philipp, HUAWEI/ZTE, or: How the CJEU Closed the Orange Book, Queen Mary Journal of Intellectual Property, vol. 6 no. 2, 2016, pp. 207 ff.

Maynard Jeff, Modular Programming, London 1972.

Mayring Philipp, Einführung in die qualitative Sozialforschung, 6th edition, Weinheim 2016, as well as 3rd edition, Weinheim 1996, if specially referred to.

McConnell Steve, Rapid Development: Taming Wild Software Schedules, Washington (DC) 1996.

McGahn Donald F., Copyright Infringement of Protected Computer Software: An Analytical Method to Determine Substantial Similarity, Rutgers Computer and Technology Law Journal, vol. 21, 1995, pp. 88 ff.

McGarry Daniel D., The Metalogicon of John Salisbury: A Twelfth-Century Defense of the Verbal and Logical Arts of the Trivium, Berkeley 1956.

Melullis Klaus-Juergen, Zur Sonderrechtsfähigkeit von Computerprogrammen, in: Ann Ch./Anders W./Dreiss U./Jestaedt B./Stauder D. (eds.), Materielles Patentrecht – Festschrift für Reiman König zum 70. Geburtstag, Cologne/Berlin/Bonn/Munich 2003, pp. 341 ff.

Merges Robert, As Many as Six Impossible Patents Before Breakfast: Property Rights for Business Concepts and Patent System Reform, Berkeley Technology Law Journal, vol. 14, 1999, pp. 577 ff.

Merges Robert/Nelson Richard, On the Complex Economics of Patent Scope, Columbia Law Review, vol. 90 no. 4, 1990, pp. 839 ff.

Merton Robert/Kendall Patricia, The Focused Interview, American Journal of Sociology, vol. 51 no. 6, 1946, pp. 541 ff.

Metzger Axel/Jaeger Till, Open Content-Lizenzen nach deutschem Recht, MMR, 2003, 431 ff.

Meuser Michael/Nagel Ulrike, Expertenwissen und Experteninterview, in: Hitzler R./Honer A./Maeder Ch. (eds.), Expertenwissen: die institutionalisierte Kompetenz zur Konstruktion von Wirklichkeit, Opladen 1994, pp. 180 ff., cited as Meuser/Nagel (1994).

Meuser Michael/Nagel Ulrike, ExpertInneninterviews – vielfach erprobt, wenig bedacht. Ein Beitrag zur Methodendiskussion, in: Bogner A./Littig B./Menz W. (eds.), Das Experteninterview. Theorie, Methode, Anwendung, 2nd edition (not included in the 3rd edition of the book), Wiesbaden 2005, pp. 71 ff., cited as Meuser/Nagel (2005).

Meuser Michael/Nagel Ulrike, Das Experteninterview – konzeptionelle Grundlagen und methodische Anlage, in: Pickel S./Pickel G./Lauth H.-J./Jahn D. (eds.), Methoden der vergleichende Politik- und Sozialwissenschaft. Neue Entwicklungen und Anwendungen, Wiesbaden 2009, pp. 465 ff., cited as Meuser/Nagel (2009).

Meuser Michael/Nagel Ulrike, Das Experteninterview – Wissenssoziologische Voraussetzungen und methodische Durchführung, in: Friebertshäuser B. (ed.), Handbuch Qualitative Forschungsmethoden in der Erziehungswissenschaft, 4th edition, Weinheim 2013, pp. 457 ff., cited as Meuser/Nagel (2013).

Mieg Harald/Brunner Beat, Experteninterviews: Reflexionen zur Methodologie und Erhebungstechnik, Schweizerische Zeitschrift für Soziologie, vol. 30 no. 2, 2004, pp. 199 ff.

Moehring Philipp, Die Schutzfähigkeit von Programmen für Datenverarbeitungsmaschinen**,** GRUR, 1967, pp. 269 ff.

Mooers Calvin N., Computer Software and Copyright, ACM Computing Survey, vol. 7 no. 1, 1975, pp. 45 ff.

Morgan Chris/Lanford David, Facts and Fallacies, London 1981.

Morse, Janice M./Field Peggy A., Nursing Research: The Application of Qualitative Approaches, Cheltenham 1995.

Mott Kelsey M., The Concept of the Small Patent in European Legal Systems and Equivalent Protection under United States Law, Virginia Law Review, vol. 49 no. 2 1963 Charlottesville, pp. 232 ff.

Millson Murray/Wilemon David, The Strategy of Managing Innovation and Technology, Upper Saddle River (NJ) 2008.

Mueller Barbara K./Oertli Reinhard, Urheberrechtsgesetz (URG), Bundesgesetz über das Urheberrecht und verwandte Schutzrechte. Mit Ausblick auf das EU-Recht, deutsches Recht, Staatsverträge und die internationale Rechtsentwicklung, 2nd edition, Berne 2012, cited as Staempfli Commentary to the Swiss CopA (editor).

Murer Angelika/Zurmuehle Sarah, Die Bedeutung digitaler Daten in der Fusionskontrolle anhand der Unternehmensübernahmen durch Google und Facebook, in: Epiney A./Kern M./Hehemann L., Schweizerisches Jahrbuch für Europarecht/Annuaire Suisse de Droit Européen 2014/2015, Zurich 2015, pp. 455 ff.

Neff Emil F./Arn Matthias, Urheberrecht im EDV-Bereich, in: David L./von Bueren R. (eds.), SIWR II/2, Basel/Geneva/Munich 1998.

Newell Allen, Response: The Models Are Broken, the Models Are Broken!, Pittsburgh University Law Review, vol. 47, Pittsburgh 1986, pp. 1023 ff.

Newman Max H. A., Alan Mathison Turing: 1912-1954, in: Biographical Memoirs of Fellows of the Royal Society, vol. 1, London 1955, pp. 253 ff.

Nimmer Melville B./Nimmer David, Nimmer on Copyright, vol. 6, New Providence (NJ) 2014, cited as Nimmer/Nimmer (2014).

Nimmer Melville B./Nimmer David, Nimmer on Copyright, vol. 1, New Providence (NJ) 2016, cited as Nimmer/Nimmer (2016).

Nimmer David/Bernacchi Richard L./Frischling Gary N., A Structured Approach to Analyzing the Substantial Similarity of Computer Software in Copyright Infringement Cases**,** Arizona State Law Review, vol. 20, 1988, pp. 625 ff.

Nordhaus William D, Theory of Innovation. An Economic Theory of Technological Change, The American Economic Review, vol. 59 no. 2, 1969, pp. 18 ff.

Noth Michael/Buehler Gregor/Thouvenin Florent, Markenschutzgesetz (MSchG), 2nd edition, Berne 2017, cited as Staempfli Commentary to the Swiss MSchG (editor).

Offut Jeff, Quality Attributes of Web Software Applications, IEEE Computer Society Press, vol. 19 no. 2, 2002, pp. 25 ff.

Ogilvie John W., Defining Computer Program Parts Under Learned Hand's Abstractions Test in Software Copyright Infringement Cases, Michigan Law Review, vol. 91 no. 3, 1992, pp. 526 ff.

Ohly Ansgar, Software und Geschäftsmethoden im Patentrecht, in: Bauknecht K./Forstmoser P./Zehnder C. A., Computer und Recht, vol. 17 no. 12, 2001, pp. 809 ff.

Osterrieth Christian, Technischer Fortschritt – eine Herausforderung für das Patentrecht? Zum Gebot der Verhältnismäßigkeit beim patentrechtlichen Unterlassungsanspruch, GRUR, 2018, pp. 985 ff.

Patton Michael Quinn, Qualitative Evaluation and Research Methods, 3rd edition, Thousand Oaks 2002.

Perelman Bruce, Proving Copyright Infringement of Computer Software: An Analytical Framework, Loyola of Los Angeles Law Review, vol. 18, 1985, pp. 919 ff.

Pfadenhauer Michaela, Das Experteninterview. Ein Gespräch zwischen Experte und Quasi-Experte, in: Bogner A./Littig B./Menz W. (eds.), Experteninterview. Theorie, Methode, Anwendungsfelder, 3rd edition, Wiesbaden 2009, pp. 99 ff.

Pfeiffer Axel, Zur Diskussion der Softwareregelungen im Patentrecht. Zum Ausschluss von „Programmen für Datenverarbeitungsanlagen… als solche" von der Patentfähigkeit, GRUR, 2003, 581 ff.

Picht Peter Georg, FRAND wars 2.0 – Rechtsprechung im Anschluss an die Huawei/ZTE-Entscheidung des EuGH (Teil 1), Wirtschaft und Wettbewerb, vol. 68 no. 5, 2018, pp. 234 ff., cited as Picht (2018a).

Picht Peter Georg, FRAND wars 2.0 – Rechtsprechung im Anschluss an die Huawei/ZTE-Entscheidung des EuGH (Teil 2), Wirtschaft und Wettbewerb, vol. 68 no. 6, 2018, pp. 300 ff., cited as Picht (2018b).

PICKEL GERT, Der Einbezug des Indificuums in die Länderanalyse – Umfrageforschung und vergleichende Politikwissenschaft, in: Pickel S./Pickel G./Lauth H.-J./Jahn D. (eds.), Methoden der vergleichende Politik- und Sozialwissenschaft. Neue Entwicklungen und Anwendungen, Wiesbaden 2009, pp. 297 ff.

PICKEL GERT/PICKEL SUSANNE, Qualitative Interviews als Verfahren des Ländervergleichs, in: Pickel S./Pickel G./Lauth H.-J./Jahn D. (eds.), Methoden der vergleichende Politik- und Sozialwissenschaft. Neue Entwicklungen und Anwendungen, Wiesbaden 2009, pp. 441 ff.

PICKEL SUSANNE, Die Triangulation als Methode in der Politikwissenschaft, in: Pickel S./Pickel G./ Lauth H.-J./Jahn D. (eds.), Methoden der vergleichende Politik- und Sozialwissenschaft. Neue Entwicklungen und Anwendungen, Wiesbaden 2009, pp. 517 ff.

PINHEIRO JOHN/LACROIX GERARD, Protecting the Look and Feel of Computer Software, High Technology Law Journal, vol. 1, 1986, pp. 411 ff.

RAGAVAN SRIVIDHYA/MURPHY BRENDAN/DAVÉ RAJ, FRAND v. Compulsory Licensing: The Lesser of the Two Evils, Duke Law & Technology Review, vol. 14 no. 1, 2016, pp. 83 ff.

RANDELL BRIAN, On Alan Turing and the Origins of Digital Computers, University of Newcastle Upon Tyne, Computing Laboratory, Newcastle 1972.

RAUBER GEORG, Der urheberrechtliche Schutz von Computerprogrammen, in: Bauknecht K./Forstmoser P./Zehnder C. A., Computer und Recht, vol. 17, Zurich 1988, cited as RAUBER (1988).

RAUBER GEORG, Inhalt und Schranken des urheberrechtlichen Softwareschutzes, in: Software-Schutz. Software-Haftung, vol. 9, Schriftenreihe SAV, Zurich 1992, pp. 33 ff., cited as RAUBER (1992).

RAUBER GEORG, Lauterkeitsrechtlicher Softwareschutz, in: Thomann F. H./Rauber G. (eds.), Softwareschutz, Bern 1998, pp. 1 ff., cited as RAUBER (1998).

REHBINDER MANFRED/VIGANÒ ADRIANO, Urheberrecht und verwandte Schutzrechte mit ausführenden Verordnungen, Nebengesetzen, zwischenstaatlichen Verträgen, 3rd edition, Zurich 2008.

REICHMAN JEROME H., Legal Hybrids Between the Patent and Copyright Paradigms, Columbia Law Review, vol. 94 no. 8, 1994, pp. 2432 ff.

RIHOUX BENOÎT, Qualitative Comparative Analysis (QCA) and Related Techniques: Recent Advances and Challenges, in: Pickel S./Pickel G./Lauth H.-J./Jahn D. (eds.), Methoden der vergleichende Politik- und Sozialwissenschaft. Neue Entwicklungen und Anwendungen, Wiesbaden 2009, pp. 365 ff.

ROSSNAGEL ALEXANDER, Rechtswissenschaftliche Technikfolgenforschung. Umrisse einer Forschungsdisziplin, Baden-Baden 1993.

Royce Winston W., Managing the Development of Large Software Systems: Concepts and Techniques, Proceedings IEEE WESCON, vol. 26 no. 8, 1970, pp. 1 ff.

Rueesch Corinna, Die Weitergabe von Standard-Software, in: Bauknecht K./Forstmoser P./Zehnder C. A., Computer und Recht, vol. 18, Zurich 1988.

Samuelson Pamela, Does Copyright Protection Under the EU Software Directive Extend to Computer Program Behavior, Languages and Interfaces?, European Intellectual Property Review, February, 2012, pp. 158 ff., cited as Samuelson (2012).

Samuelson Pamela, Is Software Patentable?, Communications of the ACM, vol. 56 no. 11, 2013, pp. 23 ff., cited as Samuelson (2013).

Samuelson Pamela, Reconceptualizing Copyright's Merger Doctrine, Journal of the Copyright Society of the U.S.A., vol. 63, 2016, pp. 417 ff., cited as Samuelson (2016).

Samuelson Pamela, Functionality and Expression in Computer Programs: Refining the Tests for Software Copyright Infringement, Berkeley Technology Law Review, vol. 31 no. 3, 2017, pp. 1215 ff., cited as Samuelson (2017a).

Samuelson Pamela, Strategies for Discerning the Boundaries of Copyright and Patent Protections, Notre Dame Law Review, vol. 92 no. 4, 2017, pp. 1493 ff., cited as Samuelson (2017b).

Samuelson Pamela/Davis Randall/Kapor Mitchell D. Reichman Jerome H., A Manifesto Concerning the Legal Protection of Computer Programs, Columbia Law Review, vol. 94, 1994, pp. 2308 ff. (Samuelson et al.)

Samuelson Pamela/Scotchmer Suzanne, The Law and Economics of Reverse Engineering, Yale Law Journal, vol. 111 no. 7, 2002, pp. 1575 ff.

Scherer Frederic M., Nordhaus' Theory of Optimal Patent Life: A Geometric Reinterpretation, The American Economic Review, vol. 62 no. 3, 1972, pp. 422 ff.

Schiffner Thomas, Open Source Software, Freie Software im deutschen Urheber- und Vertragsrecht, Munich 2003.

Schricker Gerhard/Loewenheim Ulrich, Urheberrecht Kommentar, 5th edition, Munich 2017, cited as Commentary to the German UrhG (editor).

Schwabach Aaron, Internet and the Law: Technology, Society, and Compromises, Santa Barbara 2014.

Scotchmer Suzanne, Standing on the Shoulders of Giants: Cumulative Research and the Patent Law, The Journal of Economic Perspectives, vol. 5 no. 1, 1991, pp. 29 ff., cited as Scotchmer (1991).

Scotchmer Suzanne, Innovation and Incentives, Cambridge (MA) 2006, cited as Scotchmer (2006).

Schlatter Sibylle, Der Rechtsschutz von Computerspielen, Benutzeroberflaächen und Computerkunst, in: Lehmann M. (ed.), Rechtsschutz und Verwertung von Computerprogrammen, 2nd edition, Cologne 1993, pp. 169 ff.

Schmidt Douglas C., Model-Driven Engineering, IEEE Computer Society Press, vol. 39 no. 2, 2006, pp. 25 ff.

Schoelch Guenther, Patentschutz für computerimplementierte Entwurfsmethoden – Ein Kulturbruch?, GRUR, 2006, pp. 969 ff.

Schuetz Alfred, Der gut informierte Bürger, in: Brodersen A. (ed.), Alfred Schütz. Gesammelte Aufsätze, vol. 2, The Hague 1972, pp. 85 ff.

Schuetz Fritz/Meinefeld Werner/Springer Werner/Weymann Ansgar, Grundlagentheoretische Voraussetzungen methodisch kontrollierten Fremdverstehens, in: Arbeitsgruppe Bielefelder Soziologen (eds.), Alltagswissen, Interaktion und gesellschaftliche Wirklichkeit, vol. 2, Wiesbaden 1980, pp. 433 ff., cited as Schuetz et al.

Schuhmacher Dirk, Schutz von Algorithmen für Computerprogramme, Schriften zum Informations-, Telekommunikations- und Medienrecht, vol. 27, Muenster 2004.

Schulze Gernot, Urheberrechtsschutz von Computerprogrammen – geklärte Rechtsfrage oder blosse Illusion?, GRUR, 1985, pp. 997 ff.

Schwaber Ken, Scrum Development Process, in: Sutherland J.V./Patel D./Casanave C./Miller J./Hollowell G. (eds.), Business Object Design and Implementation, London 1997, pp. 117 ff.

Schwaber Ken/Beedle Mike, Agile Software Development with Scrum, Aurora (IL) 2001.

Schwarz Claudia/Kruspig Sabine, Computerimplementierte Erfindungen – Patentschutz von Software?, 2nd edition, Cologne 2018.

Seale Clive, The Quality of Qualitative Research, Qualitative Inquiry, vol. 5 no. 4, 1999, pp. 465 ff.

Shapiro Carl, Navigating the Patent Thicket: Cross Licenses, Patent Pools, and Standard Setting, Innovation Policy and the Economy, vol. 1, 2001, pp. 119 ff.

Singer Friedemann, Programmieren in der Praxis, 2nd edition, Stuttgart 1984.

Slongo Doris, Der Softwareherstellungsvertrag, in: Bauknecht K./Forstmoser P./Zehnder C. A., Computer und Recht, vol. 21, Zurich 1991.

Smith Preston G., Accelerated Product Development: Techniques and Traps, in Kahn K. B. (ed), The PDMA Handbook of New Product Development, 2nd edition (not included in the 3rd edition of the book), Hoboken 2004, pp. 173 ff.

Sommerville Ian, Software-Engineering, 10th edition, Harlow 2016, as well as 4th edition published in 1992 in Wokingham where particularly referred to.

Spindler Gerald, Rechtsfragen der Open Source, Cologne 2004.

Staffelbach Oliver, die Dekompilierung von Computerprogrammen gemäss Art. 21 URG, Dissertation at the University of Zurich, Berne 2003.

Steinke Ines, Gütekriterien qualitativer Forschung, in: Flick U./von Kardorff E./Steinke I. (eds.): Qualitative Forschung. Ein Handbuch, 11th edition, Reinbek b. Hamburg 2015, pp. 319 ff.

Stigler Rachel, Ooey GUI: The Messy Protection of Graphical User Interfaces, Northwestern Journal of Technology and Intellectual Property, vol. 12 no. 3, 2014, pp. 215 ff.

Straub Wolfgang, Der Sourcecode von Computerprogrammen im schweizerischen Recht und in der EU-Richtlinie über den Rechtsschutz von Computerprogrammen, Archiv für Urheber- und Medienrecht, vol. III, Bern 2001, pp. 807 ff., cited as Straub (2001a).

Straub Wolfgang, Individualität als Schlüsselkriterium des Urheberrechts, GRUR Int, 2001, pp. 1 ff., cited as Straub (2001b).

Straub Wolfgang, Software im System des Immaterialgüterrechts, Jusletter, publication of 16. April 2002, cited as Straub (2002).

Straub Wolfgang, L'ingénierie inverse et la propriété intellectuelle, Zeitschrift für Schweizerisches Recht, vol. 122 no. I, 2003, pp. 3 ff., cited as Straub (2003).

Straub Wolfgang, Softwareschutz, Zurich/St. Gallen 2011, cited as Straub (2011).

Straub Wolfgang, Rechtlicher Schutz von Software-Entwicklungen, in: Weinmann C./Münch P./Herren J. (eds.), Schweizer IP-Handbuch. Intellectual Property – Konzepte, Checklisten und Musterdokumente für die Praxis, Basel 2013, pp. 287 ff., cited as Straub (2013).

Straub Wolfgang, Verträge für agil geführte Projekte, Jusletter, publication of 21. Dezember 2015, cited as Straub (2015).

Strauss Anselm, Qualitative Analysis. For Social Scientists, Cambridge (UK)/New York/Melbourne 1987/1988.

Stutz M. Robert/Beutler Stephan/Kuenzi Muriel (eds.), Designgesetz (DesG), Berne 2006, cited as Staempfli Commentary to the Swiss DesG (editor).

Swanson Guy E., Frameworks For Comparative Research: Structural Anthropology and the Theory of Action, in: Vallier I. (ed.), Comparative Methods in Sociology: Essays on Trends and Applications, Berkeley 1971, pp. 141 ff.

Swanson E. Burton/Dans Enrique, System Life Expectancy and the Maintenance Effort: Exploring their Equilibration, MIS Quarterly, vol. 24 no. 2, Minneapolis 2000, pp. 277 ff.

SwissQ, Trends und Benchmarks Studie Schweiz, Zurich/Berne 2019.

Takeuchi Hirotaka/Nonaka Ikujiro, The New New Product Development Game, Harvard Business Review, vol. 64 no. 1, Cambridge (MA) 1986, pp. 137 ff.

Tansey Oisín, Process Tracing and Elite Interviewing: A case for Non-Probability Sampling, in: Pickel S./Pickel G./Lauth H.-J./Jahn D. (eds.), Methoden der vergleichende Politik- und Sozialwissenschaft. Neue Entwicklungen und Anwendungen, Wiesbaden 2009, pp. 481 ff.

Thomann Felix H., Grundriss des Softwareschutzes, in: Bauknecht K./Forstmoser P./Zehnder C. A., Computer und Recht, vol. 24, Zurich 1992, cited as Thomann (1992).

Thomann Felix H., Softwareschutz durch das Urheberrecht, in: Thomann F. H./Rauber G. (eds.), Softwareschutz, Bern 1998, pp. 1 ff., cited as Thomann (1998).

Thouvenin Florent, Funktionale Systematisierung von Wettbewerbsrecht (UWG) und Immaterialgueterrechten, Dissertation at the University of Zurich, Zurich 2005, cited as Thouvenin (2005).

Thouvenin Florent, Offenbarung und Ausführbarkeit – Ein auseinander zu haltendes Paar House of Lords – Synthon BV v. Smithkline Beecham plc, Opinions of the Lords of Appeal for Judgment in the Cause, October 20, 2005, sic!, 2006, pp. 362 ff., cited as Thouvenin (2006).

Thouvenin Florent, Patentierung von Geschäftsmethoden und Computerprogrammen: The English Approach: Court of Appeal (Civil Division) – Aerotel Ltd. vs. Telco Holdings Ltd., Telco Global Distribution Ltd. and Telco Global Ltd., and in the matter of patent applicaton GB 0314464.9 in the name of Neal William Macrossan, both on appeal from the High Court of Justice, Chancery Division (Patents Court), October 27, 2006, sic!, 2007, pp. 664 ff., cited as Thouvenin (2007).

Thouvenin Florent, Irrtum: Je kleiner der Gesttaltungsspielraum, desto eher sind die Schutzvoraussetzungen erfüllt, in: Berger M./Macchiacchini S. (eds), Populäre Irrtümer im Urheberrecht, Festschrift für Reto M. Hilty, Zurich 2008, pp. 61 ff., cited as Thouvenin (2008a).

Thouvenin Florent, Microsoft: Offene Schnittstellen – offene Märkte? Urteil des EuGH vom 17. September 2007. Microsoft vs. Europäische Kommission (T-201/04), sic!, 2008, pp. 469 ff., cited as Thouvenin (2008b).

Thouvenin Florent, Computerimplementierte Erfindungen: Status quo im im Europäischen Patentrecht – Entscheidung der Grossen Beschwerdekammer des Europäischen Patentamtes vom 12. Mai 2010 (G 3/08), sic!, 2010, pp. 808 ff., cited as Thouvenin (2010).

Thouvenin Florent/Berger Mathis, Kapitel 6.2. Patentrecht, Kapitel 6.1. Urheberrecht, Kapitel 6.4. Wettbewerbsrecht (UWG), Kapitel 6.6. Markenrecht, in: WEKA, Informatikrecht für die Praxis (Loseblatt), Zurich 2005.

Tom Willard K./Newberg Joshua, Antitrust and Intellectual Property: From Separate Spheres to Unified Field, vol. 66, 1997, pp. 167 ff.

Troller Alois, Urheberrecht und Ontologie, UFITA, vol. 50, 1967, pp. 385 ff., cited as Troller (1967).

Troller Alois, Immaterialgüterrecht: Patentrecht, Markenrecht, Muster- und Modellrecht, Urheberrecht, Wettbewerbsrecht, vol. 1, 3rd edition, Basel 1983, cited as Troller (1983).

Troller Alois, Immaterialgüterrecht: Patentrecht, Markenrecht, Muster- und Modellrecht, Urheberrecht, Wettbewerbsrecht, vol. 2, 3rd edition, Basel 1985, cited as Troller (1985).

Ullrich Hanns, Technologieschutz nach TRIPS: Prinzipien und Probleme, GRUR Int., 1995, pp. 623 ff., cited as Ullrich (1995).

Ullrich Hanns, Lizenzkartellrecht auf dem Weg zur Mitte, GRUR Int., 1996, pp. 555 ff., cited as Ullrich (1996).

Ullrich Hanns/Heinemann Andreas, Abschnitt VII. Immaterialgüterrecht, Teil B: Die Anwendung der Wettbewerbsregeln auf die Verwertung von Schutzrechten und sonst geschützten Kenntnissen, in: Immenga U./Mestmaecker E.-J. (Eds.), Wettbewerbsrecht, vol. 1, 5th edition (no included in 6th edition), Munich 2011, pp. 1668 ff.

Ulmer Eugen, Der Urheberrechtsschutz wissenschaftlicher Werke unter besonderer Berücksichtigung der Programme elektronischer Rechenanlagen, in: Bayerische Akademie der Wissenschaften. Philosophisch Historische Klasse, Sitzungsberichte, vol. 1., Munich 1967, pp. 3 ff.

Vogel Berthold, Wenn der Eisberg zu schmelzen beginnt… – Einige Reflexionen über den Stellenwert und die Probleme des Experteninterviews in der Praxis der empirischen Sozialforschung, in: Brinkmann Ch./Deeke A./Völkel B. (eds.), Experteninterviews in der Arbeitsmarktforschung, Nuernberg 1995, pp. 73 ff.

von Bueren Roland/Meer Michael A., Der Werkbegriff, in: David L./von Bueren R. (eds.), SIWR II/1, Basel/Geneva/Munich 2014, pp. 58 ff.

Von Lewinski Silke, Der EG-Richtlinienvorschlag zur Harmonisierung der Schutzdauer im Urheber- und Leistungsschutzrecht, GRUR Int., 1992, pp 724 ff.

von Weizsaecker Carl Christian, Rechte und Verhältnisse in der modernen Wirtschaftslehre, Kyklos, vol. 34 no. 3, 1981, pp. 345 ff.

Walker Victor, Developments in the Concept of a Patentable Invention in USA and Europa: Computer Programs and Methods of Doing Business, Zurich 2001.

Waller Spencer Weber, Antitrust and Social Networking, North Carolina Law Review, no. 90, 2011, pp. 1771 ff.

Waltl Peter, Geschützte und nicht geschützte Computerprogramme, Berlin 1990.

Wandtke Artur-Axel/Bullinger Winfried, Praxiskommentar zum Urheberrechtsgesetz, 4th edition, Munich 2014, cited as Wandtke/Bullinger (editor).

Webster Jane/Watson Richard T., Analyzing the Past to Prepare for the Future: Writing a Literature Review, Management Information System Quarterly, vol. 26 no. 2, 2002, pp. xiii ff.

Welch Andreas/Mueller Christoph, Patente – Quo vadis? – Eine Erwiderung, sic!, 2002, 290 ff.

Wen Lian/Tuffley David/Rout Terry, Using Composition Trees to Model and Compare Software Process, in: O'Connor R./Rout T./McCaffery F./Dorling A. (ed.), Software Process Improvement and Capability Determination: 11th International Conference, Heidelberg 2011, pp. 1 ff.

Wiebe Andreas, Know-how-Schutz von Computersoftware: eine rechtsvergleichende Untersuchung der wettbewerbsrechtlichen Schutzmöglichkeiten in Deutschland und den USA, Munich 1993.

Wickihalder Urs, Entwicklungen im Bereich der Patentierung von computergestützten Erfindungen, sic!, 2002, pp. 579 ff.

Widmer Ursula, Der urheberrechtliche Schutz von Computerprogrammen, Zeitschrift für Schweizerisches Recht, 1993, vol. 1, pp. 247 ff.

Wirth Niklaus, Program Development by Stepwise Refinement, Communications of the ACM, vol. 26 no. 1, 1983 (reprint of 1971), pp. 70 ff.

Wittmer Hans Rudolf, Der Schutz von Computersoftware – Urheberrecht oder Sonderrecht?, in: Reh-binder M./Larese W. (eds.), Schriften zum Medienrecht, vol. 6, Berne 1981.

Woestehoff Knut, Die First Sale Doktrin und der U.S.-amerikanische Softwaremarkt, Studien zum Gewerblichen Rechtsschutz und zum Urheberrecht, vo. 47, Hamburg 2008.

Wolff Christian, Zwangslizenzen im Immaterialgüterrecht, Schriften zum deutschen und internationalen Persönlichkeits- und Immaterialgüterrecht, vol. 10, 2005 Goettingen.

Zehnder Carl August, Informatik-Projektentwicklung: Projekt, Anwendung, Nutzung, 4th edition, Zurich 2003.

Zelle Karl/Schlechtner Oswald/Schmid Franz, Algorithmen – Programme, Programmpakete: Eine Programmbibliothek für Wissenschaft und Verwaltung, Vienna/Munich 1975.

Zimmermann Philip R., Cryptography for the Internet, Scientific American, vol. 279 no. 4, 1998, pp. 110 ff.

Zirn Frank, Softwarerechtsschutz zwischen Urheberrecht und Patentrecht: Aktuelle Entwicklungen vor historischem Hintergrund und internationaler Zusammenhang, Dissertation at the Humboldt University Berlin, Stuttgart 2004.

Electronic Sources

AcclaimIP, 2015 US PATENTING STATISTICS, available at <http://www.acclaimip.com/2015-us-patenting-statistics/> (retrieved September 6, 2021).

The Agile Manifesto, published 2001, available at <http://agilemanifesto.org> (retrieved September 6, 2021).

Agrawal Shavak/Sankaran Anush/Laha Anirban/Chemmengath Saneem Ahmed/Shrivastava Disha/Sankaranarayanan Karthik, What is deemed computationally creative?, IBM Journal of Research & Development, vol. 1 no. 63, paper 3, available at <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8645678&tag=1> (retrieved September 6, 2021), cited as Agrawal et al.

Bahir Dar University, Institute of Technology, School of computing and Electrical Engineering Computer Science Program, Industrial Project I, Car Rental and Online Reservation System. For Budget Car Rent (BRC) Bahir-Dar Branch, published on February 6, 2012, available at <https://www.academia.edu/3442775/Car_Rent_and_Online_Reservation_System> (retrieved September 6, 2021), cited as Bahir Dar University.

Bridging the Gap, "functional specification", available at <http://www.bridging-the-gap.com/functional-specification/> (retrieved July September 6, 2021).

BSA-The Software Alliance, Global Software Survey 2018, available at <https://gss.bsa.org/wp-content/uploads/2018/05/2018_BSA_GSS_Report_en.pdf> (retrieved September 6, 2021), cited as BSA-The Software Alliance, Global Software Survey.

BSA-The Software Alliance, Software: A €910 Billion Catalyst for the EU Economy, 2016, available at <https://softwareimpact.bsa.org/pdf/Economic_Impact_of_Software_Report.pdf> (retrieved September 6, 2021), cited as BSA-The Software Alliance, European Union.

BSA-The Software Alliance, The $1 Trillion Economic Impact of Software in the United States, 2016, available at <https://softwareimpact.bsa.org/pdf/Economic_Impact_of_Software_Report.pdf> (retrieved September 6, 2021), cited as BSA-The Software Alliance, United States.

The Business Dictionary:

- "design thinking", available at <http://www.businessdictionary.com/definition/design-thinking.html> (retrieved July 27, 2019);
- "electronic", available at <http://www.businessdictionary.com/definition/electronic.html> (retrieved July 27, 2019);
- "system design", available at <http://www.businessdictionary.com/definition/system-design.html> (retrieved July 27, 2019);

– "time to market", available at <http://www.businessdictionary.com/definition/time-to-market.html> (retrieved July 27, 2019).

Cambridge Dictionary, "computer", available at <http://dictionary.cambridge.org/dictionary/english/computer> (retrieved July 27, 2019).

Caroli Paulo, Agile Bridge Analogy, published on July 21, 2008, available at <http://agiletips.blogspot.ch/2008/07/agile-bridge-analogy.html> (retrieved September 6, 2021).

CNET, Apple, Google lead tech takeover of top global brands, article published October 5, 2015, available at <http://www.cnet.com/news/apple-google-lead-tech-takeover-of-top-global-brands/> (retrieved September 6, 2021).

Computerhope, "update", available at <https://www.computerhope.com/jargon/u/update.htm> (retrieved September 6, 2021).

Copeland Jack, Alan Turing: The Codebreaker Who Saved 'Millions of Lives', contribution published on June 19, 2012 on BBCNews, available at <http://www.bbc.com/news/technology-18419691> (retrieved September 6, 2021).

Cornell University, Legal Information Institute, "contract", available at <https://www.law.cornell.edu/wex/contract> (retrieved September 6, 2021).

Dictionary.com, "database", available at <https://www.dictionary.com/browse/database> (retrieved September 6, 2021).

Dreier Thomas, Technik und Recht – Herausforderungen zur Gestaltung der Informationsgesellschaft, keynote speech at the award ceremony of the honorary senator dignity to Dr. h.c. Klaus TSCHIRA on November 30, 1999, available at <https://www.zar.kit.edu/DATA/veroeffentlichungen/_tschira-vortrag_8a58e13.pdf> (retrieved September 6, 2021), cited as Dreier (1999).

Foldoc Dictionary

– "information and communication technology", available at <http://foldoc.org/Information%20and%20Communication%20Technology> (retrieved September 6, 2021);

– "systems development life cycle", available at <https://foldoc.org/Systems%20Development%20Life%20Cycle> (retrieved September 6, 2021).

The History of SEO, Short History of Early Search Engines, article available at <http://www.thehistoryofseo.com/The-Industry/Short_History_of_Early_Search_Engines.aspx> (retrieved September 6, 2021).

Humble Jez, The Case for Continuous Delivery, blog-entry published on February 13, 2014 on ThoughtWorks, available at <https://www.thoughtworks.com/insights/blog/case-continuous-delivery> (retrieved September 6, 2021).

Killick James/Sakellariou Stratigoula, HUAWEI v ZTE – No More Need to Look at the Orange Book in SEP Disputes, article published on September 18, 2015 in

Competition Policy International, available at <https://www.competitionpolicyinternational.com/huawei-v-zte-no-more-need-to-look-at-the-orange-book-in-sep-disputes/> (retrieved September 6, 2021).

KUHN D.L, Selecting and Effectively Using a Computer Aided Software Engineering Tool, report no. WSRC-RP-89-483/conference no. CONF-891192-7, published in 1989, available at < http://www.osti.gov/scitech/biblio/5611931> (retrieved September 6, 2021).

LINTHICUM DAVID S., Defining the Value of Continuous Deployment for an Agile World, report published November 20, 2014 in Gigacom, available at <https://gigaom.com/report/defining-the-value-of-continuous-deployment-for-an-agile-world/> (retrieved September 6, 2021).

Macmillan Dictionary, "add-on", available at <https://www.macmillandictionary.com/dictionary/british/add-on_2> (retrieved September 6, 2021).

THE MEDIUM, System Design in Software Development, blog article published on September 24, 2018, available at <https://medium.com/the-andela-way/system-design-in-software-development-f360ce6fcbb9> (retrieved September 6, 2021).

Merriam-Webster Dictionary

- "engineering", available at <http://www.merriam-webster.com/dictionary/engineering> (retrieved September 6, 2021);
- "reverse engineering", available at <http://www.merriam-webster.com/dictionary/reverse%20engineer> (retrieved September 6, 2021);
- "software", available at <http://www.merriam-webster.com/dictionary/software> (retrieved September 6, 2021);
- "software engineering", available at <http://www.merriam-webster.com/dictionary/software-engineering> (retrieved September 6, 2021).

Motive Glossary, "Look-and-Feel", available at <http://www.motive.co.nz/glossary/-looknfeel.php> (retrieved August 4, 2019).

OECD, 2017 Statistics on Life Expectancy, available at <https://data.oecd.org/healthstat/life-expectancy-at-birth.htm> (retrieved September 6, 2021).

OUT-LAW.COM, Patents Directive killed by European Parliament, article published on July 6, 2015, available at <http://www.out-law.com/page-5884> (retrieved September 6, 2021).

OXAGILE, Waterfall Software Development Model, blog-entry of February 5, 2014, available at <http://www.oxagile.com/company/blog/the-waterfall-model/> (retrieved September 6, 2021).

PETIT NICOLAS, HUAWEIi v ZTE: Judicial Conservatism at the Patent-Antitrust Intersection, CPI Antitrust Chronicle, vol. 10 no. 2, published October 2015, available at <https://orbi.uliege.be/bitstream/2268/192440/1/Huaweï%20v.%20ZTE.pdf> (retrieved September 6, 2021)

Schwaber Ken/Sutherland Jeff, The Scrum Guide, published in 2013, 2017 edition, available at <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100> (retrieved September 6, 2021).

Gartner, Gartner Says Global IT Spending to Grow 1.1 Percent in 2019, article published on April 17, 2019, available at <https://www.gartner.com/en/newsroom/press-releases/2019-04-17-gartner-says-global-it-spending-to-grow-1-1-per-cent-i> (retrieved September 6, 2021).

The Sage Encyclopedia of Qualitative Research Methods

- "interview guide", available at <http://methods.sagepub.com/Reference/sage-encyc-qualitative-research-methods/n238.xml> (retrieved September 6, 2021);
- "quantitative research", available at <http://methods.sagepub.com/Reference/sage-encyc-qualitative-research-methods/n361.xml> (retrieved September 6, 2021).

Standish Group, The Chaos Report 2015, report published in 2015 (2018 edition not accessible), available at <https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf> (retrieved September 6, 2021).

Stanford University, Human Resources, Interview Guidelines, available at <http://staffing.stanford.edu/pdf/Interview-Guidelines.pdf> (retrieved July 27, 2019), cited as Stanford University.

Techopedia

- "agile development", available at <https://www.techopedia.com/definition/13564/agile-software-development> (retrieved September 6, 2021);
- "decompile", available at <https://www.techopedia.com/definition/16374/-decompile> (retrieved September 6, 2021);
- "high-level language", available at <https://www.techopedia.com/definition/3925/high-level-language-hll> (retrieved September 6, 2021);
- "iterative and incremental development", available at <https://www.techopedia.com/definition/25895/iterative-and-incremental-development> (retrieved September 6, 2021);
- "PC game", available at <https://www.techopedia.com/definition/31136/personal-computer-game-pc-game> (retrieved September 6, 2021);
- "Structured Systems Analysis and Design Method", available at <https://www.techopedia.com/definition/3983/structured-systems-analysis-and-design-method-ssadm> (retrieved September 6, 2021);
- "system design", available at <https://www.techopedia.com/definition/29998/system-design> (retrieved September 6, 2021).

Techtarget, "fork", available at <https://whatis.techtarget.com/definition/fork> (retrieved September 6, 2021).

Techterms, "bit", available at <http://techterms.com/definition/bit> (retrieved September 6, 2021).

U.S. Centers for Disease Control and Prevention, 2017 Statistics on Life Expectancy, available at <http://www.cdc.gov/nchs/fastats/life-expectancy.htm> (retrieved September 6, 2021).

University of Durham Department of Computer Science/Keele University Software Engineering Group of the School of Computer Science and Mathematics, Guidelines for Performing Systematic Literature Reviews in Software Engineering, version 2.3, published on July 9, 2007, available at <https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf> (retrieved September 6, 2021), cited as University of Durham/Keele University.

Wikimedia, Traffic Analysis Report – Operating Systems, published August 5, 2015, available at <https://stats.wikimedia.org/wikimedia/squids/SquidReportOperatingSystems.htm> (retrieved September 6, 2021).

Wikipedia

- "analogue technics", available at < https://en.wikipedia.org/wiki/Analogue_electronics> (retrieved September 6, 2021);
- "digital electronics", available at <https://en.wikipedia.org/wiki/Digital_electronics> (retrieved September 6, 2021);
- "server", available at <https://en.wikipedia.org/wiki/Server_(computing)#cite_note-1> (retrieved September 6, 2021);
- "software engineer", available at <https://en.wikipedia.org/wiki/Software_engineer> (retrieved September 6, 2021);
- "plug-in", available at <https://en.wikipedia.org/wiki/Plug-in_(computing)> (retrieved September 6, 2021).

Wiltgen Bryan J./Goel Ashok K., A computational theory of evaluation in creative design, IBM Journal of Research & Development, vol. 1 no. 63, paper 4, available at <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8618365> (retrieved September 6, 2021).

WIPO

- "trade secret", available at <https://www.wipo.int/tradesecrets/en/> (retrieved September 6, 2021);
- "utility models", available at <https://www.wipo.int/patents/en/topics/utility_models.html> (retrieved September 6, 2021).

Others

2nd Law Amendment Act to the UrhG (2. UrhÄndG) of June 9, 1993, BT-Drucks, 12/4022.

Administrative Council of the European Patent Office, Protocol on the Interpretation of Article 69 EPC of 5 October 1973 as revised by the Act revising the EPC of 29 November 2000, published in 2001, available at <https://www.epo.org/law-practice/legal-texts/html/epc/2016/e/ma2a.html> (retrieved September 6, 2021), cited as Protocol on the Interpretation of Article 69 EPC of 5 October 1973 as revised by the Act revising the EPC of 29 November, 2000.

DIN standard series (Normenreihe) 69901 on project management (Projektmanagement), published in January, 2009, cited as DIN.

Dispatch to the Swiss Copyright Act (Botschaft zu einem Bundesgesetz über das Urheberrecht und verwandte Schutzrechte), published June 19, 1989, BBl 1989 III 477, cited as Dispatch to the Swiss Copyright Act, BBl. 1989 III.

European Commission, Communication from the Commission — Guidance on the Commission's Enforcement Priorities in Applying Article 82 of the EC Treaty to Abusive Exclusionary Conduct by Dominant Undertakings, 2009/C 45/02, published February 24, 2009, available at <https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:52009XC0224(01)&from=EN> (retrieved September 6, 2021), cited as Guidance Paper of the European Commission on Abusive Exclusionary Conduct by Dominant Undertakings.

European Patent Office, Guidelines for Examination in the European Patent Office, November 2019 edition, available at <http://www.epo.org/law-practice/legal-texts/guidelines.html> (retrieved September 6, 2021), cited as Guidelines for Examination of the European Patent Office.

European Patent Office/Japan Patent Office/U.S. Patent and Trademark Office, Trilateral Projects, Report on Comparative Study Carried Out Under Trilateral Project 24.2 Computer-Related Inventions, published in 1997, available at <https://www.jpo.go.jp/e/news/kokusai/nichibeiou/pj242/index.html> (retrieved September 6, 2021), cited as Patent Offices Comparative Study (1997).

European Patent Office/Japan Patent Office/U.S. Patent and Trademark Office, Trilateral Projects, Report on Comparative Study Carried Out Under Trilateral Project B3b Business Method Related Inventions, published in June 2000, available at <https://www.trilateral.net/sites/default/files/attachments/8f3391ec-f272-4d1f-b78b-a50b76beec7b/main.pdf> (retrieved September 6, 2021), cited as Patent Offices Comparative Study (2000).

European Union Intellectual Property Office, Guidelines for Examination of Registered Community Designs, August 2016 edition, available at < https://euipo.europa.eu/tunnel-web/secure/webdav/guest/document_library/contentPdfs/

law_and_practice/designs_practice_manual/WP_1_2017/examination_of_applications_for_registered_community_designs_en.pdf> (retrieved September 6, 2021), cited as European Guidelines for Examination of Registered Community Designs.

ISO-, IEC- and IEEE-standard series

- IEEE 1175.2-2006 IEEE Recommended Practice for CASE Tool Interconnection – Characterization of Interconnections, published 2006;
- IEEE 828-2012 as standard for configuration management in systems and software engineering, published 2012;
- ISO/IEC 2382:2015 on information technology, vocabulary, published in 1976, revised multiple times, last revised in 2015;
- ISO/IEC 12207:2017 on software life cycle processes, published in 1995, revised in 2008 and 2017;
- ISO/IEC 19500-2:2012 on interoperability, published in 2003, revised in 2012;
- ISO/IEC TR 19759:2016 on software engineering body of knowledge (SWEBOK);
- ISO/IEC 19770:2015 on ICT asset management, published in 2013, revised in 2015;
- ISO/IEC TS 24748-1:2016 on live cycle management, published in 2010, revised in 2016 and 2018;
- ISO/IEC 25000:2014 as guide to systems and software quality requirements and evaluation, published in 2005, revised in 2014;
- ISO/IEC 25010:2011 on system and software quality models, published in 2011;
- ISO/IEC 25040:2011 on evaluation process, published in 2011;
- ISO/IEC 25051:2014 on requirements for quality of Ready to Use Software Product (RUSP) and instructions for testing, published in 2006 and revised in 2014;
- ISO/IEC TR 25060:2010 on Common Industry Format (CIF) for usability, general framework for usability-related information, published in 2010;
- ISO/IEC 26514:2008 on requirements for designers and developers of user documentation, published in 2008;
- ISO/IEC/IEEE 15288:2015 on system life cycle processes, published in 2008, revised in 2015;
- ISO/IEC/IEEE 15939:2017 on measurement process, published in 2017;
- ISO/IEC/IEEE 23026:2015 on engineering and management of websites for systems, software, and services information, published 2006, revised in 2015;
- ISO/IEC/IEEE 24765:2017 on systems and software engineering, vocabulary, published in 2010, revised in 2017;

- ISO/IEC/IEEE 26515:2011 on developing user documentation in an agile environment, published in 2011, revised in 2018;
- ISO/IEC/IEEE 42010:2011 on architecture description, published in 2007, revised in 2011.

OECD Directorate for Science, Technology and Industry, Perspective Statistical Analysis of Science, Technology and Industry Working Paper, Global Overview of Innovative Activities from the Patent Indicators, no. 3, 2006, available at <https://www.oecd-ilibrary.org/docserver/674714465672.pdf?expires=1630953582&id=id&accname=guest&checksum=3D2747DBF2D7D383B3D2130D31274FFF> (retrieved September 6, 2021), cited as Perspective STI Working Paper.

Swiss Federal Institute for Intellectual Property, Richtlinien für die Sachprüfung der nationalen Patentanmeldungen, July 2011 (version of January 2019), cited as Swiss Guidelines for the Substantive Evaluation of Patent Applications.

Swiss State Secretariat for Education, Research and Innovation Report, published in 2016, available at <https://www.usi.ch/sites/default/files/storage/attachments/press-research-innovation-switzerland-sefri.pdf> (retrieved September 6, 2021).

U.S. Commission On New Technology, Final Report on New Technology Uses of Copyrighted Works, Sections on Software Copyrights, published on July 31, 1978, available at <https://repository.jmls.edu/cgi/viewcontent.cgi?article=1573&context=jitpl> (retrieved on September 6, 2021), cited as Contu.

U.S. Congress, Office of Technology Assessment, Computer Software and Intellectual Property, Background Paper, OTA-BP-CIT-61, published in March, 1990, available at <https://ota.fas.org/reports/9009.pdf> (retrieved September 6, 2021), cited as U.S. Congress (1990).

U.S. Congress, Office of Technology Assessment, Finding a Balance: Computer Software. Intellectual Property, and the Challenge of Technological Change, report, OTA-TCT-527, published in May, 1992, available at <https://ota.fas.org/reports/9215.pdf> (retrieved September 6, 2021), cited as U.S. Congress (1992).

U.S. Department for Defense, Defense System Software Development. Military Standard, document DOD-STD-2167, published on June 4, 1985, available at <http://www.product-lifecycle-management.com/download/DOD-STD-2167A.pdf> (retrieved September 6, 2021).

U.S. Department of Justice/Federal Trade Commmission, Antitrust Guidelines for Licensing Intellectual Property, published April 6, 1995, updated January 12, 2017, available at <https://www.justice.gov/atr/IPguidelines/download> (retrieved September 6, 2021).

U.S. Patent and Trademark Office, Examination Guidelines for Obviousness of the U.S. Patent and Trademark Office, Examination Guidelines for Determining Obvious-

ness Under 35 U.S.C. 103, R-11.2013, available at <http://www.uspto.gov/web/offices/pac/mpep/s2141.html> (retrieved September 6, 2021), cited as Examination Guidelines for Obviousness of the U.S. Patent and Trademark Office.

U.S. Patent and Trademark Office, Manual of Patent Examining Procedure, January 2018 edition, available at https://www.uspto.gov/web/offices/pac/mpep/ (retrieved September 6, 2021), cited as Manual of Patent Examining Procedure of the U.S. Patent and Trademark Office.

U.S. Senate, Report of the 2nd Session of the 82nd Congress, report no. 1979, Washington 1952.

U.S. Senate Foreign Relations Committee, Report on the 39th Session of the 100th Congress, report no. 352, Washington 1988.

U.S. Supreme Court, Opinions, no. 08–964, published October, 2009, available at <http://www.supremecourt.gov/opinions/09pdf/08-964.pdf> (retrieved September 6, 2021), cited as Supreme Court Opinion.

WIPO, Model Provisions of the Expert Group on the Legal Protection of Computer Software, Geneva 1978, cited as WIPO Model Provisions on the Protection of Computer Software.

WIPO, website section on industrial rights, available at <http://www.wipo.int/designs/en/> (retrieved September 6, 2021) cited as WIPO, industrial rights.

WIPO Standing Committee on the Law of Patents, document SCP/15/3, Annex II, Computer Programs as Excluded Patentable Subject Matter, written by Brad Sherman, published on February 3, 2011, available at <http://www.wipo.int/edocs/mdocs/scp/en/scp_15/scp_15_3-annex2.pdf> (retrieved September 6, 2021), cited as WIPO Standing Committee on the Law of Patents.

List of Figures

Figure on the Title Page (Source: own illustration). III
Figure 1. The Standard Phase Model (Source: own illustration). 63
Figure 2. The Linear Development Approach (Source: own illustration). 69
Figure 3. Bridge Construction According to the Linear Development Approach (Source: own illustration). 70
Figure 4. Iterative Development (Source: own illustration). 72
Figure 5. Incremental Development (Source: own illustration). 73
Figure 6. An Example for the Structure of a Scrum Board (Source: own illustration). 74
Figure 7. Bridge Construction According to the Spiral Development Approach (Source: own illustration). 75
Figure 8. The Continuous Delivery Approach (Source: own illustration). 77
Figure 9. Bridge Construction According to the Continuous Delivery Approach (Source: own illustration). 78
Figure 10. Incremental Extension (Source: own illustration). 338
Figure 11. Different Scenarios for Inner Change (Source: own illustration). 342

All illustrations used in this dissertation were designed by Sarah Leins-Zurmuehle and drawn by Robin Leins.

Abstract

Today, the software industry is regarded as one of the most creative and dynamic industries in the world. New, innovative products are constantly being launched, and known established paths for analogue solutions are being challenged and abandoned. Sheltering software through copyright and patent law has been a major point of contention for the past 40 years. A particular difficulty lies in determining the scope of protection in intellectual property law. While the legal framework is highly standardized through several multinational codes, its practical application differs significantly among the various jurisdictions. Economists and lawyers have tried to make the present protection system more balanced and at the same time more efficient. Unfortunately, these analyses often neglect the technical realities – the practicalities and needs of software developers and right holders. The discourse is frequently limited to one particular closed discipline.

This doctoral thesis examines the rapidly changing and complex software development market and discusses some pressing legal issues. The aim is to analyse how computer programs are developed and commercialized nowadays, and to evaluate to what extent copyright and patent law are able to reflect these structures. Based on these conclusions, it is then explored what an optimal protection scope for computer programs could look like in copyright and patent law. In 12 expert interviews, technical in-house specialists were questioned about how software companies work today, how they proceed in developing their programs, how they commercialize them through sales and services, and to what extent they use legal measures to protect their software. The results of these qualitative interviews were then evaluated systematically and legally reintegrated.

The main achievement of this thesis is to provide the necessary basic scientific research regarding how the software industry works today and how this might affect copyright and patent law. From a legal perspective, it offers novel insights and points of view on existing doctrines. Further, it acknowledges some prevailing trends in the software industry which have so far been largely unaddressed by copyright and patent law. It also discusses possible approaches to how these problems could be tackled in the future.

Chapter 1: Introduction

I. Introduction to the Research Subject

At the beginning of every theory stands an idea. It rarely arrives out of the blue, but rather is induced by a personal passion, an accidental encounter, an observation or a specific problem. I believe this statement applies to both software development and my dissertation project. 1

The initial triggers for my doctoral thesis were of a primarily personal nature. Several people in my close circle expressed their legal discontent with their work in the information and communication technology (ICT) sector. They explained that they would increasingly be confronted with over-contractualization when dealing with computer programs. Particularly when working with large corporations, they would more and more have to issue contractual assurances that they had not unlawfully embedded third-party intellectual property in their products. For them this compliance guarantee represented an issue that they, as small- and medium-sized enterprises, could hardly handle. Most of them had, at least once in their career, faced an infringement of their own intellectual property (IP) and had found it extremely difficult to enforce their own rights, let alone administer third-party IP. These colleagues and also other software developers and product managers stated that they were unsatisfied with the current legal situation and that their expectations as well as some core needs were not being met. Some of them went as far as claiming that the current intellectual property regime in Switzerland would discourage development and innovation in software altogether. 2

That statement caught my ears. If this were true, it would represent a massive locational disadvantage for Switzerland. Software has become possibly the most important market good worldwide. Not only has everyday life become almost unimaginable without ICT, but it further helps us to design our businesses effectively and increase their rates of performance. The ICT industry, combining both produced goods and supplied services, has left its mark on the world's economies. According to market research by Gartner, Inc., in 2018 a total of USD 3.747 trillion was spent on software worldwide, with a further 1.1 per cent growth expected for 2019.[1] The Software Alliance BSA estimated that, in 2016 alone, software contributed a sum of USD 1.07 trillion to the value- 3

[1] GARTNER.

added GDP in the United States and EUR 910 billion in the European Union.[2] These numbers indicate that the software market is vital for economies, and that its importance is still growing. Yet software not only represents an economic top seller and is a flourishing market of its own, it also drives innovation and productivity in other economic core markets. One of the most important application areas outside the classic ICT market is the pharmaceutical and medical product industries, in particular medical technology. According to estimates of the Software Alliance BSA for 2016, around USD 52 billion in the United States and EUR 12.7 billion in the European Union were invested in research and development (R&D) of software.[3] The Swiss State Secretariat for Education estimates that Switzerland has spent approximately CHF 18.5 billion on research and development activities, suggesting that Switzerland is among the OECD's top spenders on software R&D.[4] The high investments indicate that the trend for digitalization and software is going to continue.

4 At the same time, we seem to be experiencing a global problem of intellectual property infringement in software. According to an international survey by the Software Alliance BSA, the use of unlicensed software is still widespread, accounting for up to 50 per cent or higher in the majority of all surveyed countries.[5] This international association that advocates for the interests of the software industry warns that in order to reduce software IP (particularly copyright) infringement, governments should ensure that their legal frameworks provide effective measures to protect innovation and promote the means for redress and collaboration among stakeholders.[6] Likewise, the Swiss State Secretariat for Education emphasizes that to secure a competitive advantage and adequate opportunities, companies aiming to make their inventions a success need to be able to protect their goods with strong intellectual property rights.[7] This would call for favourable framework conditions which are in tune with current developments.[8]

---

2 See 2016 statistics of BSA - the Software Alliance, a global association advocating for the software industry before governments and in the international marketplace, for the United States in: BSA-The Software Alliance, United States, 1, and for the European Union in BSA-The Software Alliance, European Union, 1.

3 For the United States, see BSA-The Software Alliance, United States; for the European Union see BSA-The Software Alliance, European Union.

4 Swiss State Secretariat for Education, 38.

5 BSA-The Software Alliance, Global Software Survey, 2.

6 BSA-The Software Alliance, Global Software Survey, 15.

7 Swiss State Secretariat for Education, 31.

8 Swiss State Secretariat for Education, 7 f.

The crossroads between intellectual property law and software has repeatedly given rise to major controversies. Today, computer programs are legally protected mainly with copyright and patent law. In the field of copyright, we see difficulties because the institute for protection of authors is static and not geared to a dynamic constantly evolving product, such as software. The latest development methods, particularly, represent an entirely new challenge for copyright. At the same time, we may experience heated debates about the general eligibility of computer programs for patent law in both Europe – where patents for computer programs are largely precluded – and the United States – where excessive over-patenting and misuse of computer patents is claimed. There is great unrest. Many stakeholders and lawyers have criticized the system of computer protection in intellectual property law.[9] A significant wave of criticism appeared in 1975, when the Association International pour la Protection de la Propriété Industrielle (AIPPI) proposed to draft a special legal system to protect software, including a *sui generis* mechanism.[10] In 1978 the WIPO then analysed legal software protection and suggested a set of Model Provisions to better capture the intellectual property of computer programs, suggesting a specially designed statute for computer programs analogous to copyright.[11] Despite various international efforts to respond to the demands and rectify the situation, in 1992 a report of the Office of Technology Assessment of the U.S. Congress (OTA) concluded that the new information system still challenged intellectual property law and that it would (still) entail numerous difficulties for current business practices and standing legal doctrines.[12] After more than four decades, international dissatisfaction has improved little and software companies and lawyers are likewise still complaining about the uncontrollable growth in legal software protection. 5

In the last couple of years, this problem appears to have become even more acute. There remain numerous uncertainties about the applicability of copyright and patent law to (new) software development and commercialization methods. To contain this problem in practice, software companies are increasingly relying on alternative protection mechanisms, for example contractual 6

9 See comment by Bill Gates during a Microsoft Innovation Day in Europe, in: Boecker, 35; Ohly, 809; Schulze 997 ff.; Harison, 113 f. and 193; economic analysis in Merges, 603 ff., see particularly 606; difficulties also addressed in U.S. Congress (1990), 5 and U.S. Congress (1992), 3 ff.; Samuelson et al., 2347 ff.; Bell/Parchomovsky, 232 f. in particular; Reichman, 2558; Hilty/Geiger (2015), 615 f.; Scotchmer (2006), 83 f.

10 AIPPI, 137.

11 No. 22 of the WIPO Model Provisions on the Protection of Computer Software.

12 See U.S. Congress (1992), 3.

obligations, greater secrecy and the use of technical protection measures. These private measures may, however, lead to unprofitable investments and increase transaction costs. In my view, it would thus be more effective to get to the root of the problem, namely copyright and patent law and find the solution here. But the question remains whether the problems raised by the software companies can be solved coherently in copyright and patent law. This leads to the next question, and the main question of my thesis, which is whether the principles in intellectual property law are still suitable for the protection of digital creations, such as computer programs. Do today's regulations cover the needs of the industry or do we need any corrective measures?

7 In this context, Rossnagel coined the concept of *jurisprudential technology design*. Law has a claim on determining behaviour, and aims to act correctively on reality, influencing developments and participating in them.[13] However, for the present thesis, I suggest a reverse approach, a *technology-driven legislation*. The aim is not the legal control of technology in the sense of creating opportunities and risks, but rather recognizing and translating factual realities in the work of software developers into law in order to reflect the actual conditions under which it has to become effective and offer practicable legal measures to protect software manufacturers. In the spirit of the quote by David Hockney mentioned on the first page of this dissertation, the aim is to gain a better understanding of the legal object to be protected. It will never be possible to make (IP) law specifically perfect for software engineering, but the goal should be to bring about an approximation to the practicalities of software development and commercialization in order to create a legal space for further creativity.

8 But before we fall into actionism to change IP software protection for the better, we first have to understand the subject matter so we can detect potential problems. Instead of unilaterally assessing technology from a purely legal perspective, this thesis aims to explore the perspective of the software developing and commercializing companies, because they are, firstly, the subject of this legislation and, secondly, the drivers of innovation and creativity in the field. Although software engineering is recognized as being of fundamental importance for our economy and for the development of future innovations, it is not yet clear what software engineers who are developing and commercializing software truly need from a legal perspective to effectively shelter their prod-

[13] For the whole abstract and more information on the concept of jurisprudential technology design, see ROSSNAGEL, 22 and 73.

ucts with IP rights. No such data is available.[14] This work will collect and analyse this missing socio-scientific qualitative data on the software companies' needs in terms of software IP protection. It will then aim to determine to what extent the current legal system, relying mainly on copyright and patent law, is able to reflect these processes and where there is room for improvement. The intention is to make useful suggestions for the legal protection of software in copyright and patent law, or provide novel legal qualifications where these are missing.

II. Definition of the Research Problem

This thesis will examine the question of how software companies today develop and commercialize computer programs, in order to establish to what extent copyright and patent law is able to reflect these structures, and how – based on these findings – computer programs could consequently be adequately protected with legal measures. 9

This is a controversial subject that has been examined by lawyers on various occasions. However, no contemporary scientific data is available to support any claims. For this reason, my dissertation represents *basic scientific research*, in which by *means of social-scientific methods* utilizable data will be collected and evaluated in order to make verifiable statements for law. This evaluation will serve as a kind of location analysis, to further the legal understanding of software engineering. It will offer new points of view on existing legal doctrines as well as novel models to legally embed newer development and commercialization approaches into the legal protection of computer programs through copyright and patent law 10

From a socio-scientific perspective, the aim of my research is to work out how software engineering companies work in practice nowadays and what requirements arise from this for the legal description of the scope of protection in copyright and patent law. The goal is to elaborate and determine the sector-specific conditions of the software industry. My research therefore involves multiple steps. The first is to learn *how software is developed and distributed to clients operationally* which involves collecting information on procedural facts and then discovering *the needs and wishes of the software developing companies*. In this context, the needs are the basic expectations of the software companies in order to function accordingly. The wishes refer to how these compa- 11

[14] Same conclusion in SAMUELSON ET AL., 2310, and MELULLIS, 355 ff.

nies want to satisfy their needs in a way that is compatible with the practices of the industry. This thesis will be documenting these needs and wishes. The results will then be *systematically reintegrated into law* in a comprehensive discussion of selected aspects in order to learn to what extent the current law is already able to capture the processes of software engineering and meet the industry's needs and wishes with regard to legal software protection, and thus find gaps that need to be closed. I also want to find out at which stage in the development process to begin with legal protection in order to make intellectual property protection as attractive and efficient as possible. Throughout the analysis, the focus will be on the legal reflection of the procedural sequences, the different work products and the underlying needs and wishes of software engineering in order to map the scope of protection closer to the actual conditions of the software industry.

12 As the title of the thesis implies, the research focus of this work lies particularly in *the creative protection scope* of software under copyright and patent law. This thesis consequently has a particular legal focus. For this analysis, *the subject matter*, *the requirements for protection* and *the terms of protection* will be examined and described in detail. There will also be comprehensive discussion on how the individual *software components* are to be classified objectively in patent law and copyright. On the basis of these features, I will suggest an improved scope of protection, which neither offers too little protection nor has an overprotective effect.

13 Although this thesis represents comprehensive basic scientific research, certain thematic *delineations* have to be made:

14 With regard to the present object of protection being examined – software – and its evident legal classification in the past, the intellectual property rights examined in this thesis will be thematically narrowed to *copyright* and *patent law*. In the context of ongoing debates, there will also be discussion on to what extent a *sui generis right*, similar to the protection of semiconductors, would be appropriate. Trademark and design law as well as aspects of contract and competition law (unfair competition and antitrust law) will only be marginally touched upon. Procedural questions will only be involved as far as they are relevant to assessing how practical the current substantive law is and at which point its evaluation can be combined with the right to protect and enforce copyright and patent law.

15 Due to its conceptual approach, this work cannot be restricted to a specific national legal order. Copyright and patent law have been strongly unified with

the help of various international treaties. They are based on common international principles, requirements and terms of use and hence exhibit a particularly universal character. However, the common criteria are interpreted independently by each jurisdiction, which in practice sometimes leads to very divergent results. This becomes particularly apparent in the case of software commercialization, as the industry operates more often across national borders. For this reason, a comparison of different national and regional solutions is crucial. Due to their strong international character, the questions to be examined would probably arise in all legal systems. However, the reference points for this doctoral thesis are primarily the international treaties and law of *the United States*,[15] of the member states of *the European Union*[16] and of *Switzerland*[17]. For the purposes of illustration, references are also made to other national legal systems.

As outlined above, this thesis concentrates on the scope of protection in patent law and copyright, with a special focus on the potential subject matter, protection requirements and the terms of protection. Second-hand use is 16

[15] The focus is mainly on *U.S. legislation and case law* because the United States, together with Japan, represents a major global player in the international software market. Many of the international working corporations are incorporated in the United States. To integrate common law into a study it is therefore important to obtain representative and sustainable market information. Furthermore, European courts and authorities tend to consider U.S. case law in their examinations, mainly because the standardized international law legislation and technical problems are comparable and U.S. jurisprudence tends to offer more long-standing and richer practice on software IP problems than European courts.

[16] In many fields of economic law, the legislation within the *European Union* has been widely unified with the help of directives and orders. The legislation and practice following from these is easily accessible and well suited to comparison with other legislation. In addition, Europe has developed its own very particular civil law practice. In order to avoid iteration, I will restrict references to available legislation to just one or two exemplary state laws for the European Union. I will particularly focus on German law, because German IP law has incorporated the E-Commerce Directive, the Copyright Directive and the Computer Program Directive, including their language, scope, purpose and systems. The German implementation of software protection law consequently represents a suitable example for the pursued doctrine within the European Union.

[17] This investigation has its origins in *Switzerland*. While there is no rich and uniform case law on the subject of software IP protection, it is becoming increasingly clear that Zurich has become an important ICT hub for the European market, which is why it is expected that this topic will also become increasingly relevant for Switzerland in the future. Switzerland is not part of the European Union, however, due to its voluntary alignment and membership of the European Patent Organisation, Swiss IP law shows many parallels to the law of the European Union and its member states.

mentioned to the extent that it is relevant to the issues involved. This thesis expressly does not cover any additional author's rights of use or moral rights, nor does it go into the topics of exhaustion and legal limitations. This would go beyond the scope of this work and require a different methodological approach to take into account the interests of third parties, such as users. It also does not cover how to legally capture data and databases or the problematic of interfaces. The methodological approach pursued would not do justice to these issues as, again, third-party perspectives would have to be considered more strongly. This work concentrates on products and services in the field of software and can only provide a specific amount of basic research in this core area.

17 The focus of the study is clearly on the practices, needs and wishes of *software companies* developing computer programs, as they are best placed to inform lawyers on how software is developed and commercialized. Although intellectual property law attempts to achieve a balance with other interests, such as those of the public or users, this is not the emphasis of the current work. This problem would be too comprehensive and could not be adequately addressed in a doctoral thesis. However, software companies and developers were found to be strongly into networking, often also being consumers of other people's software products and services. The multiple perspectives of software engineers were also visible in the interview study, so I was able to discover and embed further points of view and arguments other than just those of the professional role of the software developer.

18 Due to the thematic focus, only *software companies constituted under private law* were surveyed in the interview study. In order to factor out regulatory issues and complex conflicts of interest, public institutions and universities had to be excluded from the investigation.

19 At the beginning of my research, I focused on *user software*. As reliable trends could be observed at a very early stage of the interview study, the sample, and with it the focus of this thesis, was expanded to *developers of other types of software*, such as individual software, sector-specific software and standard software. However, this thesis does not cover the topics of file-sharing or software piracy.

20 From a *temporal perspective*, my investigation focuses on the processes of software development and commercialization, both being summable in one life cycle. According to this approach, *the development process* involves every creative and innovative step from a first idea to a final product that can be im-

plemented or offered on the market. *Commercialization* then implies distributing the software directly to the costumers. The registration of intellectual property rights such as patents as well as IP enforcement may appear under both time segments, but are not in the scope of this thesis due to their formal legal character.

Finally, it should be noted that this work focuses on the *proprietary distribution of software under classic IP law*. Approaches of freeware, Open Source and Free Software are partially studied in the interviews, but not legally reintegrated in this thesis due to the work's focus on the needs and wishes of software companies regarding legal software protection through proprietary IP and not contractual law. 21

III. Chapter Overview

This dissertation examines the scope of protection of software in copyright and patent law. This topic represents an interface of intellectual property law, technical questions concerning the development and commercialization of software products as well as economic considerations regarding project management. The methodology used, on the other hand, was derived from sociology. 22

The fundamental basics in the respective subject areas of law, computer science and project planning as well as the socio-scientific methodology are only dealt with summarily. The aim of the respective sections is not to provide a deeper dogmatic examination of the existing teaching. Rather, the reader will be given the necessary means to understand the findings of the interview study and comprehend the discussion of results. For further information on the foundation, refer to the corresponding specialist literature. A brief overview of the chapters of this doctoral thesis follows: 23

Introduction 24

This first chapter introduces the reader to the research environment of law and software. It explains in broad outline why the research problem was chosen and why an approach of basic scientific research was conducted to obtain utilizable data. In a second section, the research question is described and the field of research is delineated.

Methodology 25

In the second chapter the procedure and the encountered problems of the data collection and analysis are revealed. It first provides a general description

of the socio-scientific methodology. It then explains how and why I decided to conduct expert interviews in order to collect suitable data on the research question. I detail how I prepared myself for the interview study with an extensive literature review, how I designed the interview guidelines and how I selected the experts for the survey. I also elaborate on how the collected data was evaluated using grounded theory, and the hypotheses validated through member validation.

26 Technical Foundation
The aim of the third chapter is to introduce the reader to the technical characteristics and project work to enable comprehension of the structures in software engineering and evaluation of the extent to which the law can reflect them. The technical processes and facts are essential in order to be able to build on them later, so this chapter briefly explains the basic technical terms. Thematically, the focus is on the technical terms and procedures in development and commercialization to fully outline the research question, but economic questions in the project planning and commercialization of software projects are also briefly covered. Again, it is not the intention to offer a detailed review of the literature, but to give the reader an overview of the technical basics.

27 Status Quo of Legal Software Protection
In the fourth chapter, the reader is provided with a short overview of the relevant software property rights. Special focus is given to copyright and patent law as well as international agreements in these fields. Associated fields of law such as contract and competition law as well as other intellectual property rights are briefly listed, but are not dealt with in detail due to the scope of this thesis. The legal description focuses on the object of protection, the requirements for protection and the terms of protection in copyright and patent law.

28 Findings of the Interview Series
The fifth chapter presents the results of the twelve interviews conducted. First, it describes how software is developed by companies today, how this process is structured, what the most common development methods are and how a software project is set up. A further section explains how software companies offer their products and services on the market and how they anticipate the software life cycle. The chapter then discusses concepts important to intellectual property such as knowledge transfer, innovation and creativity. A major focus of this chapter is the interviewees' analysis of the current software IP regime and its potential for improvement. This chapter ends with the topics of IP infringement and enforcement.

Discussion of Selected Problems 29

The sixth chapter reflects the results of the interview study in legal terms. In particular, it examines to what extent the existing law can reflect the structures in the development and distribution of computer programs. It discusses whether it makes sense to continue to legally protect software with a hybrid of copyright and patent law, or just one of the two, or to use a *sui generis* legal institution instead. The existing provisions on the object of protection, the requirements and terms of protection are discussed in more detail and are critically questioned. Finally, the problems of second-hand works and standard essential creations and the potential for incremental improvements are explored within a short excursus to demarcate the scope of protection. This chapter addresses possibilities for improvement and points of criticism, and proposes different solutions as well as rough models which could better integrate the needs and wishes of the interviewed software developing companies and improve the current legal protection of computer programs with copyright and patent law.

Prospect and Closing 30

Finally, in the seventh chapter, the implications of the findings and their legal integration are reflected. It is transparently explained to what extent the results of this thesis permit substantial conclusions and where further research would be necessary. The chapter notes briefly where the existing copyright and patent law exhibits the potential to better encompass the needs and wishes of software companies with the help of a contemporary interpretation or a change in practice. Where the structures in software development and commercialization cannot be effectively addressed with the current copyright and patent law, and a revision in law would be necessary, concrete proposals for regulatory adaptations are offered.

Chapter 2: Methodology

31 The previous chapter described the research question this thesis will investigate. This second chapter introduces the methodology used to conduct the research. I discuss the methods considered and the reasons for those selected. I also provide insights into how I structured the research project and the difficulties encountered. First, there is a brief introduction to the socio-scientific field and its methology. I then explain how I relied on an extensive literature review to gain an overview of the subject and to reflect on how the literature discusses the research question. I outline why I decided to use expert interviews to collect data and how these were prepared in order to achieve credible and representative results. The collected data were processed through partial transcription and analysed using a systematic coding method called grounded theory. Finally, I describe how the findings of the analysis were used to construct general theories that could subsequently be used to compare the current legal structure with the practicalities of software development.

32 A particular difficulty in the research project was the fact that, with software engineering, innovation management and intellectual property law, my thesis involved three broadly varying thematic areas. Consequently, there were not only legal questions but also economic, technical and sociological ones and it became very difficult to fulfil the demands of each discipline. I tried to overcome this problem by following advice provided by Hart for the researcher to remain extra flexible and focus on being open-minded whenever several disciplines are involved in the same research project. Hart believes that by pointing out to the reader what was done and how it was achieved the implications and potential difficulties of the research project are made transparent.[18] I therefore share some of these difficulties in the relevant sections of this chapter.

I. Introduction to the Socio-Scientific Approach and Selection of a Research Method

33 For many lawyers, socio-scientific research represents a black box they rarely ever get in touch with. This included the author of this thesis until this dissertation project. The following section therefore presents a short introduction to the socio-scientific area.

---

[18] See HART, 10 f., 18 and 83.

The main task of researching is to collect data, analyse it and develop a theory based on the findings. It is a process of fitting data, linking and attributing it and uncovering what was previously unrecognized. Data in this context is a collective term for all kinds of information that can be analysed following a scientific method.[19] The method followed is largely dependent on the research question and the researcher conducting the study. It is his or her job to design and apply a creative approach, ask the correct questions and generate solid knowledge from what has been observed.[20] 34

The aim of this dissertation project is to learn how companies develop and commercialize software today and how their needs and wishes can be integrated into the current legal IP system. There are several approaches that could be followed to get to know-how software is developed and commercialized. This information is part of the sector-specific know-how of engineering and project development. We need access to this knowledge in order to obtain reasonable findings and draw credible and representative conclusions. As I am no software developer myself, I had to look for third parties who were experts in their field and were willing to participate in the research. 35

Giddens once said that experts are the *guardians of formulaic truth*.[21] What he possibly meant by this is that experts possess valuable knowledge that is not easily accessible and cannot simply be learned. Rather, they have collected their knowledge and cultivated it. It is a fact that a researcher may acquire some of a research object's knowledge in time but will not be able to have the full experience and know-how of a real software developer. Therefore, the aim of this project was to disclose the practical knowledge and perceptions of software engineers that were relevant to the research question. 36

Unlocking information requires selecting an adequate methodical approach. The core of the research question concerns a legal problem. It may therefore be expected that one would rely on a legal methodology toolbox, such as the Sociology of Law. *Unfortunately, law does not itself provide a method for data collection in the practical field.* Therefore, we have to search for a method in another field of science. 37

Social sciences, including sociology and social anthropology, differentiate between qualitative and quantitative research: 38

19 Pickel/Pickel, 443, fn. 12.

20 Corbin/Strauss, 42 f. and 52, enhancing the context in Morse/Field, 125 f.

21 Giddens, 450 f.

Qualitative research involves "an interpretative, naturalistic approach to its subject matter". Qualitative methods study things in their natural settings and try to interpret the phenomena perceived from the meaning people bring to them.[22] Similarly to hermeneutics, qualitative researchers try to obtain an in-depth understanding of the research subject. And as the subject may only be observed in its primary natural setting, usually smaller samples are required to develop and prove a theory.

Quantitative methods, on the other hand, refer to a systematic empirical investigation via mathematical, statistical or simply numerical data to express a quantitative relationship between a hypothesis or proof and its subject.[23] The approach is deductive-nomological, meaning that every assumption is based on a causal nexus with a linear development.[24] Because in quantitative research one tries to find a theory that is applicable for a set of situations, usually a bigger sample has to be used.

39 While quantitative research focuses on causal determination, prediction and generalization of findings, qualitative methods try to illuminate and understand a particular situation or relationship.[25] Decisive for choosing either a quantitative or qualitative approach is the area of study and the research question being assessed. In the present case, the goal is to gain a deeper understanding of the software development process. The main focus is on the knowledge and perceptions of the developer companies. This involves the whole development and commercialization process and its legal comprehension within intellectual property law. As only very little legal evaluation of this question exists, it would be difficult to follow a deductive approach alone. Instead, we are trying to understand our research subject and gain in-depth comprehension to evaluate whether the current legal framework matches it. A process or a perception is difficult to express in quantitative data, as it is descriptive in nature.[26] Further observations, descriptions and legal integration of the research field are required. Working statistically therefore cannot provide the insights needed to understand the issue. Consequently, a qualitative method was chosen for this research project.

---

22 DENZIN/LINCOLN, 2 (not included in the 2018 edition of the book).

23 The Sage Encyclopedia of Qualitative Research Methods, "quantitative research", available at <http://methods.sagepub.com/Reference/sage-encyc-qualitative-research-methods/n361.xml> (retrieved September 6, 2021).

24 See also PICKEL/PICKEL, 442.

25 HOEPFL, 48.

26 See also PICKEL, 303, and SWANSON, 142.

Socio-scientists differentiate between three main categories of qualitative methods: 40

Observation is the creation of a "systematic description" of events, behaviours and objects in a particular social setting.[27] The aim is to relive with your own senses the same situation that the subject of observation experiences. In the context of this thesis this would mean that I would need to integrate into the daily life of software engineers and try to relive their usual courses of action, including the creative composition and inventive development. Unfortunately, designing and finalizing software involves many cognitive processes that cannot simply be experienced sensually from the outside. We want the full picture from the first thought processes to fixing them in a concept and realizing them in a definitive form. We are therefore reliant on information only the software engineers themselves can provide. A lay person cannot experience the same depth of creativity and systematic knowledge as an ordinary software developer can with a well-founded technical background and personal experience. Therefore, observation would be the wrong method for this thesis. 41

Document analysis refers to the method of evaluating documents and electronic materials in order to elicit their meaning, to gain understanding and to develop empirical knowledge.[28] This includes case studies of documents and literature reviews. The analytic procedure includes finding, selecting and assessing the relevant documents as well as synthesising data contained in those documents.[29] As described later, for the present thesis a facilitated type of literature review was used to get a first impression of the research field, but it was not used to gather data. I focused on gaining basic technical knowledge and getting acquainted with the relevant terms and processes. Sometimes with theoretical descriptions in scientific papers it is quicker to integrate newer trends of a practice and discuss them on a scientific level. The literature therefore opened new possible paths to explore for my thesis. Analysing documents further helped with the general structure of the studied fields as well as revealing knowledge gaps in the theoretical literature. I then used these insights to arrange the results of the interview series and to cross-check the findings, increasing the strength of the acquired evidence and showing possible weaknesses of the theory. Therefore, I combined the literature review with another qualitative method to approach my research question. 42

27 Marshall/Rossman, 143.

28 Bowen, 27, with further references.

29 Bowen, 28.

43 Finally, there are *interviews*. With this method researchers try to get access to real-world settings which we can't experience ourselves. A narrator enables one to gain information or knowledge that it is difficult to access from the outside.[30] As software development involves a lot of personal background and invisible cognitive processes, interviews provide a suitable method of accessing this hidden knowledge that researchers cannot experience themselves. The interviews thus provided the data I was looking for. By discovering the systems behind the feedback, deeper knowledge and practices in software development could be disclosed. Furthermore, interviews allow a researcher to gather information about various different aspects or characteristics of a research subject. This means that within the same interview, I could approach and explore several areas of interest. A systematic approach in the interviews then helped me to generalize the discovered information in order to achieve representative theories.[31]

44 I therefore decided to work with socio-scientific qualitative methods, as the legal methodological toolbox itself did not provide the required methods to gather data from a practical field. I utilized literature analysis to receive access to the study field and get an initial view of the theoretical comprehension of the topic, which helped to corroborate later findings.[32] Interviews were then used to collect data from the field. Thus, I combined the literature review and interviews to investigate my research question, conducting a *methodological triangulation*.[33]

II. Literature Review

45 A literature review is a particular form of document analysis. It may be defined as the "selection of available documents (…) on the topic, which contain information, ideas, data and evidence written from a particular standpoint, to fulfil certain aims or express certain views on the nature of the topic and how it is to be investigated, and the effective evaluation of these documents in relation to the research being proposed".[34] The literature review can help to uncover

30 Patton, 341; Bogner/Menz (2009b), 8; Gordon, 1.

31 See also Pickel/Pickel, 447 ff.

32 Bowen, 28.

33 See for full description: Pickel; Tansey, 484; Bohnsack/Geimer/Meuser, 235 ff.

34 Hart, 11.

the current state of knowledge, the common methodology used by other researchers in the field and elucidate which problems require further attention and which are already saturated.[35]

Webster and Watson recommend following a clear systematic approach in order to limit the literature to those few items that are potentially relevant and representative for your particular research question.[36] This approach gives the researcher security and provides him or her with a structured procedure that enables credible and qualitatively rich extracts. The structured process involves organizing a review, specifying the potential research question, creating a review protocol, assessing this protocol, identifying the available research, selecting the relevant studies and documents, evaluating the quality and representativeness of the selected literature, and extracting relevant data and monitoring it.[37] 46

For this research project, the literature review represented the starting point that helped me to become immersed in the field of study and familiar with the topic. I chose to work with a structured approach similar to the Systematic Literature Review model as outlined in the Guidelines for Performing Systematic Literature Review by the Universities of Durham and Keele,[38] but then slightly adapted it, because the main purpose was to get an initial idea of the relevant problems for the research question. The principal source of data of this project would be the interview series. I tried to determine the prevailing doctrine in the different fields researched. Searching online for newer contributions and reviewing the leading journals in the mentioned fields, I found a variety of potentially pertinent information. The documents were then arranged by the relevance of the content and credibility of the authors, starting with the established and representative authors. Each piece of literature was analysed for its quality, credibility and representiveness in order to guarantee a qualitatively rich literary base on which to build my further research. I then followed-up with the sources used by these authors, which allowed further relevant contributions to be accessed for analysis. When a citation chain ended, the same procedure was repeated from the beginning with a new piece of literary work. After a time, I got a feel for what could be included and what could be discarded because, with each contribution I read, the relevant ques- 47

35 Webster/Watson, xiii; see also Hart, 2 f.;

36 See Webster/Watson, xix & xv f.

37 See University of Durham/Keele University, 6 ff.

38 University of Durham/Keele University.

tions became more saturated and the arguments easier to assess. At the same time, I tried to stay open-minded for new inputs by regularly reflecting on the collected literature.

48 I spent approximately five months reading solely about technical development, software engineering and innovation management. Another five months were spent reading about the existing socio-scientific methodologies available. It took approximately six months to review the relevant literature on the legal comprehension of the subject. I focused on literary contributions that discussed similar, related or partial questions of my research field. This analysis provided the theoretical foundation and offered an understanding of the software engineering industry and innovation management. By discovering the state of the art, it was possible to narrow the scope of this doctoral project and focus on specific key issues. In total, I read over 400 literary contributions, of which nearly 200 were incorporated in this work. It is noteworthy that in the field of law and social sciences, I found and worked most with monographs and commentaries, rather than with papers or online sources. At the same time, it became apparent that the classical monographs were often outdated for the dynamic field of IT sciences. For this reason, I based my research for the technical questions almost exclusively on papers and online contributions.

49 The insights from the literature review were then used to select a suitable research method for the data collection and analysis. The extracted sections also served to isolate potential interview questions for the interview series. At the same time, the literature was incorporated into the theoretical foundation that introduces the relevant technical and legal background. Finally, the results of the literature review were used to discuss the findings and proposed solutions.

50 Although the literature review was a tedious process, it was vital for my research project. This structured and systematic procedure for the literature review helped me to feel comfortable with the study field and to select extracts which were relevant and representative.

III. Interview Series

51 After the literature review had identified the critical questions and gaps in the existing knowledge, qualitative interviews were used to gather further information on the practical needs and wishes of software engineers in developing and commercializing their computer programs. First, I looked at the focus and purpose of the interviews in order to reflect what I wanted to accomplish with them. I then reviewed the different forms of interviews and analysed what

their potential contribution and effect could be. I decided to work with so-called expert interviews and thus had to consider which experts to use for this project. The results of this evaluation were used to develop a potential sample. Finally, the content of the interviews was framed with interview guidelines to determine the structure of the process and design appropriate questions.

A. Purpose of the Interview Series

I first evaluated and reflected on the exact purpose of the interviews, and through this, the objective of the research was further elaborated. 52

Bogner and Menz differentiate between three different purposes of interviews.[39] An interview may be *explorative*, which means that it offers orientation in an entirely new or yet undiscovered field of research. It may serve to accommodate a research environment and to learn more about which factors influence a particular subject and what effects can be expected.[40] Further, an interview may have a *systematizing effect* by revealing knowledge that is difficult to access. By gathering information in a continuous and systematic approach, comparability of the results can be achieved.[41] Finally, an interview may be used to *generate theories*. Based on the subjective replies of the interviewed parties, the researcher looks for rich patterns in the interviewees' responses and decisions in order to generate generizable theoretical concepts about the particular problem or field investigated.[42] 53

For this research project, a total of 12 interviews were used, to pursue all the above purposes. Most of the information gathered about the technical and procedural aspects of software development were already established in the software engineering industry. Here, the interviews helped to organize and systematize the available knowledge for non-technicians, generating a mirror image of the current software development process. This information could then be further processed and evaluated from a legal perspective. On the other hand, some of the information on software development was entirely new, less established or not yet analysed. The interview series helped to explore undiscovered ground and identify basic structures and characteristics. This also involved finding out the personal experiences of the individual software devel- 54

[39] Bogner/Menz (2009a), 64 ff., referring to Vogel, Deeke and an earlier print of Meuser/Nagel (2005).

[40] Bogner/Menz (2009a), 64.

[41] Bogner/Menz (2009a), 64 f.

[42] Bogner/Menz (2009a), 66 f.

oping companies with the current legal framework. This question has rarely been analysed previously on the basis of qualitative data. The interviews thus had an explorative characteristic as they helped to discover new relations and factors. Finally, the main goal of the interviews was to survey individual software developing companies and obtain a generalizable pattern of how software is produced today and the difficulties encountered in protecting their goods. This data could then be used to develop theoretical concepts about the needs and wants of the involved parties in an intellectual property framework. The interviews therefore served the purposes of systematizing, exploration and theory-generating.

B. Focus of the Interviews

55 Before selecting an appropriate interview approach and designing the interview guidelines, I needed to know what kind of information I was looking for and what type of knowledge I wanted to address and have shared.

56 Through expert interviews different kinds of knowledge can be obtained. Bogner and Menz differentiate between three areas of knowledge: technical knowledge, procedural knowledge and interpretive knowledge.[43] *Technical knowledge* implies discipline-specific expertise and is usually directly accessible in an interview. *Procedural knowledge* is concerned with how particular processes work and how they can be affected. Both types of knowledge can theoretically be obtained from an interviewee's functional context alone. *Interpretive knowledge*, however, goes beyond this functional component. It depends on the interviewee's personal views and priority ranking.[44] At this point, the personal background becomes transparent and the experience and opinions of an interviewee can be sensed.

57 The present research question touches all three types of knowledge. I wanted to discover how software products could be protected adequately with intellectual property law. For this, it is important to learn how software is actually developed and the personal experiences of the affected group. An understanding and involvement with technical knowledge are important to define what can be considered as predetermined, due to the technical environment the software works in, and what the technicity involves. By this we can discover what is expected and what goes on beyond this threshold. On the other hand,

43 Bogner/Menz (2009a), 72 ff.

44 For the whole abstract see Bogner/Menz (2009a), 70 f.; also discussed in Meuser/Nagel (2009), 470.

it is important to understand how the process of software development continues ordinarily, how it is structured and what can affect it. The procedural know-how integrates the practical issues of the software market that affect the development process and lie beyond technical questions, such as the demands and behaviour of clients, the economic and technical life cycle of a product, and legal modalities. This information is hidden in the personal experiences and domestic know-how of the interviewee. And it is interpretive knowledge that is reflected when an interviewee describes what they consider to be creative, inventive or trivial or if they share with the interviewer what they consider to be an appropriate legal system. Consequently, all three areas of knowledge are touched upon, although not all to the same extent.

The information obtained can generally be grouped into either *business knowledge* or *contextual know-how*. The first type refers to information about the structures and relations of a person's daily activities. The second term is linked to particular processes or incidents experts are involved in but for which they do not have direct responsibility.[45] In the present research, the focus is on business knowledge, referring to the conditions that are involved in creating and commercializing software as well as the personal contribution a developer or company can offer. This is the generalizable information that lies within the sphere of influence of the interviewee. The knowledge sought is therefore closely associated with a high degree of responsibility for the process and a depth of knowledge of the processes and structures involved. Particular incidents may serve as a source of information for individual cases but are difficult to generalize, as they are not representative enough. Contextual know-how therefore does not meet the demands of this project. 58

The research project therefore aims to uncover the knowledge of interviewees who have a high level of responsibility for software development and commercialization. The knowledge important for this study may be defined as business knowledge that consists of technical, procedural and interpretative elements. 59

C. Selecting an Interview Type

Where information cannot be observed from the outside, an interview may serve as a kind of intermediary between the technical processes in question and the researcher. There are several types of interviews available in socio-science, varying in terms of the medium (oral to written), quantity (one-to-one 60

[45] Meuser/Nagel (2009), 470 ff.

or wide survey), and the investigative spectrum (general to specific).[46] Each interview form has a different research aim. An individually designed interview allows the researcher to get the specific type of information they require.

61 In this research project, so-called *expert interviews* – sometimes also referred to as *elite interviews* – were used. In this interview form, the focus is on talking to selected people who are able to provide some kind of *specialist knowledge* relevant to the research question.[47] Expert interviews have three main purposes: first, they can illustrate what a certain set of people think about a particular issue; second, they can be used to evaluate a larger population's characteristics, perceptions or decisions; and third, they can serve to verify what was established from other sources. Interviews are always used to reconstruct an event.[48] Expert interviews in particular reflect existing knowledge that is relayed through an audio-visual or literary medium.

62 This method offers several benefits. One major advantage of expert interviews is that they are able to gather first-hand data about processes being investigated, allowing the researcher to get accounts from a direct witness.[49] This is essential for a research field such as software engineering, where the information is not readily available to the researcher. In addition, Meuser and Nagel note that this form of interview is common for those in a professional context.[50] It offers a fast, effective and efficient procedure to gather information about experience-based knowledge.[51] One main advantage of interviews therefore is that a researcher is able to gather information about a professional environment without remaining for long in the observed setting and without influencing or disturbing the investigated field.[52] Another advantage is that expert interviews have a great political force as they are very practical for knowledge discovery; findings can potentially be used to effectively modernize older economic or political systems,[53] for example by offering scientific recommen-

46 For example an interview that focuses on a narrator is called a narrative interview. For more information, see Pickel/Pickel, 446 f.

47 Helfferich, 162; Meuser/Nagel (1994), 181 f.; Bohnsack/Geimer/Meuser, 76 ff.

48 Tansey, 484; Pfadenhauer, 99.

49 Tansey, 485.

50 Meuser/Nagel (2013), 457 ff.

51 Mieg/Brunner, 199; Deeke, 9.

52 The participating expert will, therefore, be more willing to participate in the study.

53 Bogner/Menz (2009b), 9 f.

dations for regulatory measures or jurisdiction. For this research project, the expert interviews may hence provide support and legitimacy to the acquired data.[54]

The expert interviews were used to unlock special knowledge about software development and commercialization that would not otherwise have been accessible. They served to gather rich data qualitatively, directly from people working in the software industry. This helped me to achieve a fuller picture of the current software developing industry and of problems that might occur from a legal perspective. Expert interviews therefore represented an optimal approach for my research subject. 63

D. The 'Expert' Term

Expert interviews focus on interview candidates that possess some kind of special knowledge. In a first step it therefore had to be determined which people might provide credible and relevant information and might therefore be considered as experts for this research project. 64

The term 'expert' implies a lot of different meanings. One main aspect of expert interviews is that the potential expert knows something about an environment that is not accessible for everybody, particularly for the researcher.[55] The expert in question should be suitable for the research model. The term 'expert' therefore has to be newly constructed and rethought for every new project.[56] The more accurate and specific the definition of the expert is, the more guidance can be obtained from it. 65

The available definitions for an expert are widely similar. One of the most common definitions is provided by Meuser and Nagel, two well-known European researchers in the field of expert interviews. They define an expert as a person who is responsible for the design, implementation and control of a problem-solving process.[57] Although this may appear rather simplistic at first, it covers a lot of different aspects that are important for the position of an expert. First, through defining an expert by his or her area of operation, the expert is partially reduced to his/her *functionality*. The definition also implies that, if a person is responsible for the designing, implementation and monitoring of a process, a minimum threshold of knowledge is provided. If a person is able to 66

54 Bogner/Menz (2009b), 9 f.

55 Meuser/Nagel (2013), 457 ff.

56 See also Bogner/Menz (2009a), 70 f.

57 Meuser/Nagel (2005), 73.

influence all these steps, there is a high probability that this person possesses a well-founded knowledge of the associated factors. Further, the processes of designing, implementation and control may involve several different types of knowledge – technical, procedural and interpretative knowledge – at the same time.[58] Meuser and Nagel's definition therefore illustrates the potential complexity expert knowledge may involve. When relying on Meuser and Nagel's description of an expert, the functional responsibility of the interviewee becomes more important. But as the present research question contains technical and legal aspects as well as project management, an optimal expert should have responsibility for all these areas, not only for functional technical areas. In bigger companies especially, where we can observe a tendency to assign fewer management tasks to one particular person and instead spread responsibility among a larger number of employees, selecting a perfect interviewee who meets all the requirements becomes more difficult.

67 Mayring, on the other hand, offers a *voluntaristic definition*. He believes that the personal qualities of an expert are decisive, including the particular information, skills or other qualities he or she possesses and which support them in fulfilling their tasks.[59] When looking at the field of software development, the focus would consequently be on the personal qualifications of an individual project manager, software engineer or legal counsel. Although a candidate's occupation, career and personal background can have a great effect on the know-how available, this definition seems to concentrate too much on the individual interviewee and their personal opinion instead of that of the company for whom they work. As the present research project seeks to portray not only the opinion of independent software developers with a specific background but also the position of businesses, the expert term should be broader.

68 The term 'expert' can also be determined by focusing on the *scientific work* to be accomplished. Benoit and Wiesenhomeier suggest that experts are simply people who possess a comprehensive and authoritative knowledge of the subject in question.[60] This definition focuses on the particular research project. No requirements are set out for the functional perspective of a potential interview candidate, but it implies that the expert should be determined on the basis of the particular research question. Although this definition provides the researcher with the desired flexibility, it did not help me, as a lay person, with

58 For the different types of knowledge, see above N 56.

59 Mayring, 3rd edition, 49; discussion in: Bogner/Menz (2009a), 67 f.

60 Benoit/Wiesenhomeier, 501.

any guidance on what a potential expert for this research project might look like. Although some aspects of the scientific definition were useful for the present research, it was not sufficient on its own.

Finally, the *sociology of knowledge* regards experts as a tank of specified know-how.[61] Contrary to the voluntaristic approach of Mayring, it is irrelevant how an expert has achieved his/her knowledge. Instead, the main quality of an expert is that he or she is a keeper and cultivator of knowledge. The sociology of knowledge consequently focuses on the substance that is available. By focusing only on the knowledge available, the person interviewed becomes interchangeable. Furthermore, the interpretative knowledge manifested in the personal opinion of an interviewee seems to be largely disregarded. Overall, however, the definition offered by the sociology of knowledge offers some factors that were of interest for my project. 69

All of the above definitions cover particular elements that are relevant for this research project. A perfect expert should, for example, have responsibility and be able to influence his or her field of work, and this is partially affected by his or her personal background and qualities. At the same time, what can be considered as an expert for this project is determined to a large extent by the research question. And as the main aim of this project was to learn more about software development, participating in the knowledge that a representative of the industry was willing to share was very important. None of the available definitions, alone, were able to provide an ideal picture of an expert for this project. 70

It was concluded that a multi-layered definition would therefore be more suitable. Guidance was obtained from a *constructivist* approach that differentiates between a *methodical-relational* and a *social-representational* expert function. The methodical-relational element suggests that every expert is only an expert because of somebody else's research interest. The expert therefore expresses a theoretical construct where that person has a particular type of knowledge considered relevant for the research project in question.[62] He or she becomes important because they have what the researcher is particularly looking for. Deeke notes that through this the expert-term becomes reliant on the research object in question.[63] Often, the object of study and the expert are "rela- 71

61 Meuser/Nagel (2009), 469 f.

62 Bogner/Menz (2009a), 68, referring to Meuser/Nagel (2013) and Deeke.

63 Deeke, 7.

tional" because the researcher helps to define the relevant people[64] On the other hand, the *social-representative* element implies that the expert function can only be attributed to people who are also accepted as an expert in their particular social reality. This means that the expert has to be a credible and established representative within their group of specialists.[65] We are therefore looking for candidates who are respected for their professional qualities and have a high reputation in their particular sector. It is important for a research project to base its assumptions on credible and representative findings. Finding appropriate interview candidates is therefore vital. As it can be difficult to determine who may be considered an expert, their professional reputation will help to distinguish between unsuitable candidates and potentially representative ones. It is likely that a candidate will have relevant knowledge of the field of study, if he or she is established and accepted. At the same time, being representative implies that similar qualities can be observed in other software developers. It is thus important to look for candidates that show common qualities among developers although the particular characteristics of a developer or product may vary. Otherwise, the findings may not be reproducible and utilizable, making it difficult to build a generalizable theory on them.

72 Thus, reflection on what could be assumed by the term expert represented an important step in this research project. It helped to find a uniform line for the whole project and ensure that only parties were involved that could really contribute something. This meant a reasonable use of the available resources, and guaranteed qualitatively richer findings.

73 The constructivist approach therefore built a supportive base for understanding who could be considered as an expert for this project. It was important to look for people who were able to provide an extensive scope of answers to the interview questions. The perfect candidate therefore would have knowledge about technical, procedural and legal issues while having enough experience to be able to provide an interpretive personal analysis of the problem.

E. Sampling

74 In the above section, I outlined the type of potential expert who would be suitable for my particular research project. This expert description then had to be applied to a sample group who could participate in the interviews and generate credible and representative data.

---

64 Meuser/Nagel (2005), 73.

65 Bogner/Menz (2009a), 68 f.; Meuser/Nagel (1994), 181.

It should be noted that in this research project finding appropriate and representative interview candidates took up a lot of my time. As noted by Pickel and Pickel, and also Goldstein, a major difficulty with interviews lies in getting access to potential candidates.[66] About fifty per cent of the preparatory work consisted of getting in touch with people who could provide useful contacts, and winning their confidence. Only where this confidence could be established, were appointments scheduled and the interviewees could be properly prepared for the interview situation. Arranging a sample, in practice, involves a lot of obstacles and requires good resource management. 75

1. Initial and Theoretical Sampling

The people that a researcher wants to engage in his or her studies have to be selected in a justifiable way.[67] A representative sample group is picked from the total possible candidates. This process is called sampling. Unlike quantitative methods, the choice of a sample in qualitative research is usually not statistically based. Qualitative research instead tries to generalize findings found in a particular situation in order to provide a better understanding of it.[68] By determining a pattern from individual examples, the researcher is able to obtain a potential picture of the wider situation in question. 76

In the literature, two main sampling techniques are described for qualitative social sciences which distinguish between initial sampling – where to start – and theoretical sampling – where to go.[69] *Initial sampling* seeks to find a logical starting point. It tells you which group, situation or population to investigate before you actually do so.[70] *Theoretical sampling* then carries the researcher from the first series of interviews to further ones.[71] The relevant subjects that arise during analysis of the first interviews lead to new potential interviewees. Sampling is therefore conducted at least twice: once before starting the research and once after the analysis of the first few interviews. For this research project an initial sampling was done that provided some experts 77

66 Goldstein described in his book that 'getting the interview' represents one of the major challenges a researcher faces as the interviewees have other priorities than doing additional work beside their regular tasks (Goldstein, 669). See also Pickel/Pickel, 448. See also Deeke, 16 f.; Vogel, 77 f.

67 Pickel/Pickel, 447 f.

68 Helfferich, 173; Kawulich, para. 1.

69 Charmaz, 100 ff.

70 Charmaz, 100 f.

71 Charmaz, 99 f.

to interview at the beginning. After the analysis of these first interviews, I was able to get a good impression of what the data might look like. Through theoretical sampling I then looked for other potential candidates who could provide particular information for specific problems. This process was repeated until all the relevant data categories were saturated. As the data has to be analysed at the same time as further interviews are being conducted, the researcher has to work in a circular way, integrating initial findings into the working research design.[72]

78 Theoretical sampling has the effect that the final number of candidates involved in a study only emerges after the first interviews have been held and the data have been analysed. Data saturation occurs when no new categories evolve and instead the same ones reoccur, the same stories are told. At this point no further interviews have to be held.[73] Consequently, when working with theoretical sampling and expert interviews, the researcher is often busy holding interviews and evaluating them at the same time. The number of comparable study cases as well as research resources is often restricted.[74] Qualitative work therefore makes selection by outcome, only integrating cases that actually fulfil the anticipated sample characteristics.[75,76] In qualitative research it is therefore common to work with a smaller number of cases, if the obtained data are processed diligently.[77] For this reason, theoretical sampling is sometimes criticized as being arbitrary and unscientific. The main criticism lies in the fact that, in the absence of statistic development of the sample, there is a higher risk that the sample is not representative enough and the findings thus become unreliable. Charmaz counters that qualitative research does not serve to observe the distribution of data nor does it aim to find universal findings that are applicable to every possible scenario.[78] Instead, the research aims to gain a deeper understanding of a particular situation and observe common

---

72 This circular hermeneutic approach differs from the classic quantitative one, where the researcher usually works linearly; here the researcher follows only one step at a time and usually doesn't return to any of the steps (Pickel/Pickel, 445). For further information, see also Corbin/Strauss, 135 f. and 240.

73 Corbin/Strauss, 139 f.

74 Ebbinghaus, 203 f.; Rihoux, 366.

75 If analysing their features, only inevitable conditions can be identified. Any further elements influencing a situation cannot be determined (Ebbinghaus, 206).

76 Braumueller and Goertz describe this state using the formula 'X is a necessary condition for Y, if Y is always present when X occurs' (Braumoeller/Goertz, 846).

77 Ebbinghaus, 206 ff.

78 Charmaz, 101 f.

characteristics of it. In the present case, we want to learn how software is usually developed today. Special circumstances may therefore not be illustrated appropriately but the data still suffice to draw a general picture of the interviewed companies that is transferable (or generalizable) to other ones in the same field.

To obtain credible and generalizable findings, the researcher should focus on obtaining and maintaining a high quality sample.[79] Further, the research design should be regularly reassessed and updated based on what is observed during the interview series.[80] Both aspects are included in theoretical sampling. As the initial sample remains flexible, constant reflection and reintegration of previous findings is encouraged. In this case, I started with three interview candidates who were selected through initial sampling. These three interviewees were heads of very small companies. As these directors were responsible for economic, legal and technical questions within software development, a broad first impression could be gained with only a few interviews. With the help of this initial data analysis, the first relevant categories were discovered, leading to a small adaptation of the original research design. From then on, the research design was reviewed after every two interviews, and through theoretical sampling particular candidates were selected in terms of what they could contribute to the research project. This included the type of candidate, for example, from a smaller or larger company, as well as the function of the candidate within the company. After seven interviews no new categories evolved, and the relevant categories were fully saturated after ten interviews. I then decided to hold two further interviews in order to ascertain that my findings were correct and that no further relevant information could be gathered. In total, I conducted twelve interviews with representatives of the software industry. 79

2. The Sample Group

According to Goldstein, more important than the number is the careful selection of your interviewees.[81] In order to ensure comparability of the results, each candidate should be assessed on whether he or she fulfils the predetermined requirements.[82] These requirements should reflect the common char- 80

79 Helfferich, 172.

80 Erlandson/Harris/Skipper/Allen, 153.

81 The aim of sampling is to find balanced and unbiased information sources, see Goldstein, 671 f.

82 See for more information: Ebbinghaus, 205 f.; Glaser/Strauss, 47 ff.

acteristics of software development, so that most of people working in this field can relate to the findings. We are therefore looking for cogent representatives.

81 The experts selected for this project were candidates with the necessary knowledge about technical, procedural and legal aspects in software development.[83] This definition also correlated with those the U.S. Congress describes as stakeholders in the software debate.[84] The potential candidates also needed to show a large amount of professional experience so that they could provide an interpretive personal analysis. With the help of the literature review, the basic definition of an expert was further elaborated with particular characteristics in order to determine what the potential candidates should be like. The focus was on people that had well-founded knowledge about software development, its commercialization and the legal issues associated with it. At the same time, the candidates had to be representative in their field and show a certain amount of expertise and know-how, which meant that only candidates who had been working in the ICT sector for several years were included. All potential experts had to show that they had appropriate qualifications associated with the study field, e.g. a technical degree for the position of a software engineer or a law degree with a specialized additional degree in innovation management. I hence looked for candidates that could be regarded as *stakeholders of software development and management*.

82 In a first set of explorative and informal talks with software developers, we discussed the basic structure of a software developing company. I learned that the smaller a company is, the broader the sphere of responsibility for each person working for this company. While in a two-person company both usually share the technical, management and legal responsibilities, in larger companies these remits are often assigned to particular people or departments. Here, the *management board* usually defines the objectives of a project. It also controls and organizes the employee structure and assigns the resources. The *technical employees* and *project managers* are responsible for advancing and realizing a project. The *policy and legal department* of a firm assesses what legal measures should be taken to protect and commercialize the product and how the company communicates with the public. The relevant personnel for this research project were, on the one hand, the software engineers that develop the software and have the technical and procedural knowledge, and on the other, the managers and legal counsels responsible for the rights manage-

[83] For more information, see above N 64 ff. and N 73 in particular.

[84] See U.S. Congress (1992), 8-11.

ment and commercialization of the project. They have knowledge about how the resources are invested in a development project and where the difficulties in rights management lie from a practical perspective. Their experience with the current legal system and their point of view was consequently of interest for the present study. The sample for this project was therefore narrowed to a group of experts that represented stakeholders in the software development process. This included people with a high degree of responsibility for either the project management, engineering or rights management processes. The focus was on project managers, software engineers and representatives of the management tier or legal counsels, including policy makers.

83 Cowan and Jonard note that there are three places where inventions are developed: in institutions such as universities, in profit-seeking companies and with individual inventors.[85] The present study focused on the representatives of profit-seeking companies. The findings are thus mainly applicable to this field. However, in informal talks with representatives of important technical universities very similar needs and difficulties were addressed. The findings of this study may therefore be transferable to associated groups involved in software development, such as universities and individual developers, especially as most software developers to some extent follow a profit-seeking model.[86] For practical reasons, however, the present research had to be limited to profit-seeking companies.

84 It is also considered what kind of software was developed within a company. The product range is rather large, varying from client-server software models to specially designed solutions for a particular companies and web-based codes of agencies for common user software. Each type has a different purpose and meets a particular client group's needs. At the beginning of my research, I focused on user software. As reliable trends could be observed at a very early stage of the interview study, the sample, and with it the focus of this thesis, was expanded to developers of other types of software, such as individual software, sector-specific software and standard software. However, the sample used was intentionally limited to companies that develop software autonomously and also commercialize their products to third parties. Companies that merely made use of their software in-house were therefore not part of my research, as these companies were not faced with the difficulties of intellectual property law.

85 Cowan/Jonard, 513.

86 This is particularly significant for technical institutions that often follow both models of profit-seeking and non-profit to some extent.

85 The sample was further restricted to the territories of the research question. Consequently, only software developing companies within Switzerland, the European Union and the United States were interviewed.

86 To conclude, in this research project I worked with theoretical sampling where the candidates could be chosen step by step rather then selecting all the interviewees before starting the process. By using this approach, the research model could be constantly adapted to the newly developing findings. Stakeholders in software development such as software engineers, legal counsels and management representatives were chosen as the relevant sample group for my research question. In order to achieve a broad field of development sectors, suppliers of different types of software as well as companies of different sizes were included. The sample was then further narrowed by focusing only on profit-orientated developers that produced and commercialized software to third parties within the study area of Switzerland, the European Union and the United States.

F. Preparing the Interviews

87 Having established the type of knowledge required, the method for collecting the data and the appropriate sample to work with, the last step in the interview preparation was determining the structure and content of the interviews.

1. Anticipating the Interview Situation

88 Before formulating the interview questions, I first had to become familiar with the research surroundings. For this, I needed to learn more about the field of interest and the methodology.

89 First, I studied the working environment of software developers. Schuetz describes this phase as getting into the "same relevance zone" as the research subject.[87] The aim is to become immersed in the field of study and understand the common language as well as the determining factors influencing it.[88] This was accomplished using the literature review and some informal preparatory talks. Through recognizing the relevant structures in the research field, it became easier to find the right problems to investigate and to formulate more

[87] SCHUETZ, 90 ff.

[88] MAYRING, 13 f.; PICKEL/PICKEL, 445; SCHUETZ, 90 ff.; CICOUREL, 49 ff.; PFADENHAUER, 104 f.

accurate questions. As I got to learn the relevant technical terms and understand the most important procedures, the answers from the interviews could be better classified and evaluated.

Based on these insights, the actual interview procedure could then be structured. A large amount of literature is available on what an interview situation should look like, what factors should be considered and what is important. In the interview situation, the interviewer can, for example, present him- or herself as a lay person, a co-expert of the same discipline or as an expert of another discipline.[89] Depending on how the interviewer starts and what knowledge he or she possesses of the study field, the interviewees will react and behave differently. I decided to present myself as a lawyer with a big interest in technology and project development during the preparatory correspondence with the interviewees and during the actual interview situation. I tried to express that I was no software development expert but that I had acquired a basic concept of the process and technical questions involved for my dissertation project. The interviewees usually started by explaining everything in detail. Through mirroring and integrating the obtained answers, I tried to signal to them where further explanations were required and what was clear. Bogner and Menz emphasize that the interviewee needs to feel comfortable and respected in the interview situation, and that he or she should not believe that they have to "defend their culture".[90] Girtler and Pickel emphasize that a situation should be favoured in which information can be exchanged on a professional and reasonable level.[91] I therefore tried to create an amenable interview situation and refrain from posing oppressive questions. Meuser and Nagel in this context refer to what they call a "methodically controlled other-awareness" ("methodisch kontrolliertes Fremdverstehen"). The interviewer tries to consciously discern the opposite and at the same time assimilate the interview strategy to the research interest.[92] This meant that during the interview situations, I had to be fully present in mind, process the emerging answers and adapt the interview guidelines and style continually. I tried to maintain the conversation, guiding the interviewee through the interview. Through this, I was able to react to unforseeable events and turn them into valuable contributions for my data collection. 90

89 See for example Pickel/Pickel, 454.

90 Bogner/Menz (2009a), 77 ff, particularly 81; Vogel, 80.

91 For managing prejudice and other potential disruptive influences in an interview situation, see Girtler, 186 ff., and Pickel, 302.

92 Meuser/Nagel (2005), 71 f., referring to Schuetz et al.; see also Pickel/Pickel, 454.

91 At the same time, it was important for my preparation to read about the experiences of other researchers and learn that interview situations could also fail.[93] I tried to anticipate such situations by expanding my personal methodological toolbox and trying to anticipate certain problems by knowing exactly who I was talking to and where potential difficulties could lie. Preparing myself for the particular interview situation and knowing what to expect therefore became very important. Due to careful preparation, all twelve interviews passed without any problems. I was fortunate to find twelve interviewees that were happy and willing to participate and share their insights on the current technical, economic and legal situation. They were all very well prepared and could provide valuable insights for my research question.

2. Structure and Content of the Interviews

92 It is my understanding that it is the researcher's job to formulate adequate questions that result in qualitatively rich data. The interview should guide the interviewee along the common theme and provide possibilities for the expert to fill the answers with useful knowledge.

93 There are no standardized patterns for interviews, or restrictions on how the feedback should be obtained in qualitative research. This provides the interviewer with a lot of flexibility to adapt the interview to the special needs of the project. On the other hand, it carries the risk that the interview could develop into an unsystematic question-and-answer-game. According to Corbin and Strauss, research is "a process of conjecture and verification, of correction and modification, of suggestion and defense".[94] This means that the researcher has to consistently deal with the subject and reflect on what they have observed. Therefore, it is important to think through how the interview should be before holding it. Well-planned and systematic interviews are encouraged where the questions are pre-formulated. Bohnsack, Geimer and Meuser caution that every researcher should consider how much of the interview will be standardized or structured.[95] Fully standardized questions may influence answers of the interviewee and restrict the scope of possible feedback.[96] Non-standardized or non-structured questions may instead stimulate the expert to talk on

[93] The expert can block questions, lack sufficient competence or may not want to help the researcher but rather dish the dirt about internal matters or private issues etc. See for example Meuser/Nagel (2005), 78 f.; Vogel, 80.

[94] Corbin/Strauss, 239, enhancing the context of Morse/Field, 125 f.

[95] See Bohnsack/Geimer/Meuser, 151 f. and 169 f.

[96] See also Cicourel, 100 ff.; Merton/Kendall, 546; Deeke, 19 f.

any topic of their choice. In this case, the interviewer gives away his or her control over the topic. Between full and no standardization lies the use of standardized, pre-formulated questions. For this, an interview guideline is designed. The most important questions are prepared before holding the interviews and the whole interview is shaped and structured.[97]

I therefore followed the standardized or structured approach and used interview guidelines for further back-up with a method I had not yet applied. Interview guidelines[98] are a list of semi-structured questions that cover various topics and areas an interviewer wants to explore in the interview.[99] The advantage of using interview guidelines is that it helps to limit possible "interaction effects" which otherwise could compromise the outcome of an interview.[100] Consistent guidelines further support objectivity and thus assure reproducibility and trustworthiness of the data.[101] The guide here works as a systematic and consistent base which can then be used to compare and evaluate the obtained data. Another positive effect of interview guidelines is that they can be developed closely with the research field. The researcher has full autonomy on what topics to cover and can also steer how pre-formulated the questions are and how much leeway is offered to the interviewees to give feedback. 94

For the structure of the guidelines, I tried to follow Helfferich's standards.[102] She developed what she calls the SSPS system. It is an established system to structure and organize interview guidelines. It builds on selecting potential questions ("sammeln"), reevaluating the list of collected questions under the criterion of openness, including erasing the suboptimal questions and reformulating the suggestive or too broad questions ("prüfen"), arranging the remaining questions into a logical order ("sortieren"), and finally grouping the sets of questions or bundles under a key or heading question that appears the 95

97 For more information on non-standardized and fully standardized questions, see: Atteslander, 134 ff. and 144 ff.; Pickel/Pickel, 442 and 446 ff.

98 Also referred to as: interview guide, interviewing guide, interviewing guidelines or aide memoire.

99 The Sage Encyclopedia of Qualitative Research, "interview guide", available at <http://methods.sagepub.com/Reference/sage-encyc-qualitative-research-methods/n238.xml> (retrieved September 6, 2021); Vogel, 74.

100 Meuser/Nagel (2005), 77 ff.; Littig rates those interaction effects as qualitively valuable as they could contribute in a constructive and productive manner to a positive outcome of the research (Littig, 125 ff., with further references).

101 Merton/Kendall, 546 f. and 548; Meuser/Nagel (2005), 81.

102 See Helfferich, 178 ff.

most simple. The following questions are then subsumed under this key question ("subsumieren"). The base for my interview guidelines was built from my insights from the literature review and the informal preliminary talks with software developers. The interview questions were constructed around the strategic concepts of the research project.[103] Different cases and topics were arranged in an order that would mirror the natural flow of conversation. For this purpose accessible interview guidelines were observed and analysed online.[104] I tried to limit thematic changes to the minimum. The main goal was to obtain a lucid and manageable questionnaire tool that offered enough narrative space for the experts to answer. The final interview guidelines were discussed with the project supervisors and also reviewed by a software engineer to avoid overextending the later interviewees. During the running interview phase, which took six months, the interview guidelines were reviewed and adapted several times in response to observations made as well as newer discoveries. The thematic questions regarding 'Competition in Software Engineering' were particularly difficult to articulate. Formulating clear, non-confusing open questions therefore represented one of the major challenges in the project.

96 Once the outer structure and thematic blocks of the interview guidelines were built, the individual questions had to be edited and formulated. I decided to work solely with pre-formulated questions, consequently a lot of attention had to be put into the wording. I followed the key concepts of Hopf and Helfferich to build the questions, which were formulated, open, linguistically comprehensible and assignable to the context. No questions that were judgemental or too personal were used. The language was designed to be as simple as possible, without using technical jargon.[105] For example, when I wanted to learn more about the procedure of software development from the first idea to the final product, naturally there were some particular answers I anticipated due to the literature review I had carried out previously, e.g. that the process could be split into different phases. But in order to avoid influencing the answer, the interviewee was simply asked how they developed software and how they proceeded when they started a new project.

97 In the first phase of the interview, I generally provided some basic information about the project, the confidentiality terms and the interview process so that the candidates knew what was going to happen and received an overview of

103 For more information, see Meuser/Nagel (2005), 82.

104 See for example: Stanford University, or University of Durham/Keele University.

105 See Hopf, 108; see also interpretation in: Helfferich, 107 f.

the relevant research questions. According to Liebold and Trincek, the first few questions should motivate the expert to share his or her knowledge and present themselves.[106] I thus used some kick-off questions to involve the experts, asking them about their occupation and career. The personal information gave them a gentle start while gathering information about them for later data analysis and to build connections between the various research topics.[107] The interview guidelines then led to the thematic blocks of 'software development', 'software commercialization and rights management' and 'infringements and legal disputes' which I could then elaborate on step-by-step. And as the interviewees always received a copy of the interview guidelines in advance, they could prepare themselves and knew what was coming up.

Structuring and organizing the interview guidelines represented a time-consuming part of the research project. They had to follow a logical order and favour a good flow of discussion. The questions had to be formulated in a way that was appealing for the interviewees and encouraged them to share their knowledge willingly with me as their interviewer. At the same time, the interview guidelines offered security and ensured that the interviews, which were conducted in English and German, followed scientific principles so the findings remained comparable. 98

IV. Transcription

With the consent of the interviewees, the interviews were recorded and then transcribed on the computer in hours of work into written form. Most of the 12 interviews were conducted with German-speaking experts and some were held in English. I transcribed in the respective interview language.[108] 99

Theoretically, transcription offers a complete text version of the verbal statements. In consultation with my supervisor, I instead decided to transcribe only parts, although still the majority, of the interviews, leaving out certain sections, e.g. if an interviewee gave several examples for one argument. This is called *selective transcription*.[109] After transcribing about seven interviews and analysing them, I became more comfortable with the method and started to 100

106 Liebold/Trincek, 35.

107 For more information, see Pickel/Pickel, 446 f.; Bohnsack/Geimer/Meuser, 135 f.

108 As described subsequently in N 109 ff., the interviews were then all analysed and coded in English. Relevant text passages from the interviews were occasionally translated into English in order to be able to quote them in my thesis.

109 See also discussion in Strauss, 266 f.

paraphrase certain evident statements, e.g. when the interviewees mentioned a specific point repeatedly without providing new information. The challenge in this situation was to preserve the meaning and terminology of the expert's original statement.[110] The core of the arguments and the majority of all statements were retained, only irrelevant sections were skipped in transcription. I therefore tried to follow Mayring's guidelines, which say that the more relevant the information is, the denser and more elaborated the transcription should be.[111] I therefore always processed most of each conversation.

101 The transcriptions were accompanied by a short memo with information about the interview situation, who was interviewed, what kind of company the interviewee worked for, and whether something particular happened during the interview.[112] Through this, all relevant information about the interview situation and candidates was maintained. Whenever the interview was interrupted, I took notes that were later integrated into the transcript. These notes served to qualify the information and to create representative relations between them. The transcriptions and associated memos built the base for the following data analysis.

V. Data Analysis

102 Data analysis represents the final step after data collection and transcription. The following sections describe how an adequate method for data analysis was selected and how it was applied in order to build theories.

A. Selecting a Method

103 Data analysis comes after the transcription and forms the third part of scientific research. The process usually follows three main steps: description, interpretation and generalization.[113] Through the description, the individual properties of a particular research subject are elaborated. The interpretation phase consists of a circular process, involving collecting data, gaining knowledge from their evaluation and applying this knowledge again to gather new data.

110 See the same problem described in Meuser and Nagel who explicitly allow paraphrasing, as long as the original terminology and meaning are guaranteed (MEUSER/NAGEL [2005], 84).

111 MAYRING, 95.

112 See recommendations in: PICKEL/PICKEL, 448 and 555.

113 See MAYRING, 19; PICKEL/PICKEL, 445.

During the generalization phase, the researcher attempts to build an inductive theory based on the data evaluated.[114] The theory should then give further references, for example for a possible future software IP law framework.

To obtain a theory, the collected data need to be systematically processed and elaborated in context. Glaser and Strauss emphasize that "generating a theory from data means that most hypotheses and concepts not only come from the data, but are systematically worked out in relation to the data during the course of research".[115] When the first batch of data have been analysed, these findings are systematically reintegrated into the process of collecting new data, e.g. in the interviews. The researcher can identify similarities as well as differences in the statements obtained.[116] The similarities serve to identify patterns, while the differences and particularities help to delimit them. In time, the data become more saturated through repetition of data collection and data analysis, and the patterns become more reliable.[117] The information can then be interpreted and advanced into a theory. 104

The prerequisite for systematic data analysis is the selection of an appropriate analysis method. This helps to diminish the potential negative effects of the researcher's subjective interpretation.[118] The methods for data analysis generally vary from the methods used to collect the data. However, both methods should represent a fitting match, as only a harmonious procedure can produce rich and utilizable data.[119] As with theoretical sampling, data analysis starts before the interview phase is completely finished, it is important to decide at this stage which method of data analysis will be used. 105

The analysis method should provide the researcher with a suitable structure. I started by comparing other researchers' projects, especially ones that made use of expert interviews. I observed that frequently expert interviews were combined with a data analysis method called grounded theory.[120] This is prob- 106

114 For a full description of the phases, see MAYRING, 35 f. and 36.

115 GLASER/STRAUSS, 1.

116 MEUSER/NAGEL (2005), 80; CORBIN/STRAUSS, 50.

117 MEUSER/NAGEL (2005), 80; FINCH, 215 f.; CHARMAZ, 113 ff.

118 In data analysis, the researcher has to select and determine one of multiple meanings of an incident. As there is no universal interpretation of a situation, data analysis remains subjective to some extent. For more information on this topic, see DENZIN, 322; CORBIN/STRAUSS, 64 f. with further references; CICOUREL, 76 ff.

119 PICKEL/PICKEL, 448 f.

120 Despite its misleading name, grounded theory is not a theory or hypothesis, but a method of data analysis.

ably rooted in the fact that grounded theory by definition is designed to examine phenomena involving people that are closely involved with them.[121] Grounded theory is a particular inductive method that enables systematic and structured data analysis, a critical reflection of the obtained findings which favours building theories and demarcating them at the same time.[122] The method was elaborated by Corbin and Strauss and is described in their famous textbook *The Discovery of Grounded Theory*. The book also provides helpful recommendations for people studying the method. Grounded theory as a method is well established not only in social sciences, but also in other disciplines such as information management and economics.[123] The book has been revised several times and has picked up various trends in the research field. It has been further substantiated and improved with the help of a number of dissertations and articles.

107 In their book, Corbin and Strauss describe how their method encourages circular research with theoretical sampling. The idea of developing every finding directly from your own data in the form of transcribed texts and associated background information is convincing. The simple framework ensures systematic evaluation and fosters inter-subjectivity and comprehensible findings that could be reproduced if necessary.[124] The methodical approach of grounded theory is not only established in the field of expert interviews, but is also suited to the planned circular procedure.

108 There are many other methods available that could be used to analyse data. For example, Andrews offers a method that is very similar to grounded theory. Both approaches mainly work with semi-structured interviews based on interview guides, they both gather data by theoretical sampling and both of them offer a hypothesis through generalization.[125] However, Andrews' approach does not differentiate between the different types of coding, while Corbin and Strauss emphasize and explain the approaches. Furthermore, before the collaboration with Corbin, Strauss introduced an early version of grounded theory that focused on the interpretation of expert interviews. Compared with the later version of grounded theory with Corbin, Strauss's solo approach provided a process that could be adapted more flexibly to the particular research

121 See FINCH, 214.

122 MAYRING, 103 ff.; FINCH, 219 f., 220 and 223; BOHNSACK/GEIMER/MEUSER, 97 ff.

123 For various applications of grounded theory see for example GALAL (information management) or LANGLEY (economics) for a good overview.

124 PICKEL/PICKEL, 449 f.

125 See FINCH, 223 and 225, for an informative analysis of ANDREWS.

situation.[126] Unfortunately, the analysis procedure is difficult to follow for a lay person. The new grounded theory was instead built up neatly and comprehensibly which helps the researcher to organize and prepare data analysis carefully. I reviewed many other analysis methods but none of them could provide the same qualities as grounded theory. The unique features of the method, combined with its support in the socio-scientific community convinced me to work with the data analysis method of grounded theory.

B. Analysis and Theory-Building

Grounded theory provides a clear procedure that guides the researcher through data analysis in a comprehensive and systematic way.[127] The strong structure supports the researcher in becoming aware of possible positive and negative influences that may occur. 109

The procedure consists of four main steps:[128] 110

1. Open coding: The data are collected, analysed and put into categories.
2. Comparative Analysis/Axial Coding: Systematic relationships are formed between the categories that have emerged from step one. The observed relationships are described in a more detailed manner by using coding properties to describe paradigms.
3. Selective Coding/Integration: One core category[129] is chosen and all the other categories are systematically arranged around it. This network of categories helps to form various concepts, resulting in the formulation of a so-called hypothesis.[130]
4. Conditional Matrix: The hypothesis gains a conditional dimension by adding a simple cause/effect structure, which links the various concepts to each other in order to build a substantive or formal theory.

126 Strauss, particularly 22 ff.

127 Finch, 213 and 216; Hess/Wilde, 282.

128 For a full description, see Corbin/Strauss and its description in: Mayring, 103 ff, 106 f. in particular.

129 A core category is a special class which has the potential for great explanatory relevance and offers possibilities to link other categories to it. For this it should be sufficiently abstract and appear frequently in the data. For further information, see Corbin/Strauss, 188 f.

130 In the terminology of Corbin and Strauss and grounded theory, a "hypothesis" does not represent an untested assumption a researcher formulates at the beginning of their research. Instead, it relates to the concepts the researcher builds during data analysis.

111 The bass for the evaluation in this research are the transcripts and memos from the expert interviews. The process starts by *analysing* the content of the transcripts. According to Girtler and Pickel & Pickel, it is essential to classify every response critically and refrain from integrating the answers into patterns without reflection.[131] In this research, one major challenge in analysing the expert interviews was to differentiate between a candidate's professional expertise and his or her personal opinion. During analysis, I tried to distinguish between expertise and opinions, individual approaches and the position of the company they represented. For this purpose, I used George and Bennett's verification questions to assess delicate cases and reflect on how they could be evaluated.[132]

112 The actual procedure starts with the processing of the data. Similar statements and paraphrases are marked and assembled under the same collective heading.[133] This process is called open coding; segments of data are grouped and receive a common category designation that reflects their content.[134] In this research project, I used the same set of categories for all the interviews, assigning every relevant statement of each expert to one or more categories.[135] There are several ways that such categories can be built. The researcher can either wait until the categories develop from the data, or they can make use of so-called borrowed categories by utilizing key terms mentioned in specialist literature.[136] As I conducted a broad literature review before I started with the interview series, I got to know certain categories other authors had used in their socio-scientific studies in the field of software engineering. However, the categories emerged immediately during the transcription of the first interviews and became even stronger when the first set of interviews were analysed. As I processed the individual interviews several times, I got to know which topics could be considered relevant for the analysis. Furthermore, as the interview questions discussed specific topics of interest within the research question, it was clear that some of these topics would reappear during the data analysis, although sometimes in a different context. The categories could therefore be well developed. Each category was then given a heading. For this, I tried to use terms the experts had used in their discussions.

131 For more information about this topic, see GIRTLER, 188 f., and PICKEL/PICKEL, 449 f.

132 They were "who is speaking?", "who are they speaking to?", "for what purpose are they speaking?" and "under what circumstances?" (GEORGE/BENNETT, 99).

133 MEUSER/NAGEL (2005), 85.

134 CHARMAZ, 43 ff, particularly 45.

135 This procedure was recommended in: MEUSER/NAGEL (2005), 85 f.

136 GLASER/STRAUSS, 37.

After open coding, sociological conceptualization takes place where the researcher dissociates themself from the original wording and tries to apply his/her own wording and systems to the available categories. A notional concentration is done, where common typologies and features are connected.[137] The data hereby is made sharper, increasing the scientific quality of the analysis.[138] By analysing more and more interviews, the observed categories become richer and denser in their composition. After every new interview, the data are compared to the findings obtained previously. This approach is called *comparative analysis*. By comparing each statement with the previously analysed ones, similarities and discrepancies can be observed. The similarities and differences are integrated by adding so-called properties to categories, which describe a particular aspect of the category. For example, if the category 'life cycle' discusses how long an expected life cycle of software is, the property 'technology' could describe how the better a system environment corresponds with newer technologies, the longer the life cycle is. Another property may contain the estimated duration of a potential life cycle with different numbers. The properties can show further dimensions of the categories. If the statement analysed corresponds with previous categories or properties, the statement gets the same code. If the statement differs from the previously observed categories and properties, a new code is added and it is defined as either a category or property.[139] For this project, a total of 33 categories were used, each referring to a thematic block. The categories consisted of between 2 and 60 properties. Altogether, the 12 interviews involved 1,888 code decisions. 113

When theoretical sampling is used, the circular process from data collection to analyis continues until *saturation* is reached and all the categories are fully developed and rich in content.[140] This means that the researcher has achieved a level where further data collection would not produce new aspects for the categories.[141] At this stage, the pattern of the data becomes visible and the researcher can *integrate* the categories by describing their relationships to each other.[142] The question of when the categories are fully saturated is really difficult to answer as it depends on one's personal evaluation. Strauss and Corbin stress how important it is to experience as much of the complexity as possible, 114

137 Meuser/Nagel (2005), 88 f.

138 Strauss, 25 f. and 35 f.

139 For the whole paragraph, see Langley, 700 f.; Corbin/Strauss, 93 ff., 239 and 240 f.; Bohnsack/Geimer/Meuser, 130 f.

140 Helfferich, 174 f.; Corbin/Strauss, 134 f.

141 Pickel/Pickel, 445.

142 Corbin/Strauss, 197 ff. and 295 ff.

but capturing every detail would be virtually impossible. Every study has its own limitations.[143] Charmaz suggests that research studies with a smaller scope may achieve saturation faster than the ones that want to describe a bigger story.[144] He believes that a smaller number of cases may suffice if the research question is being investigated for the first time. A bigger number is necessary if the findings contradict previous findings or want to build a formal theory. In my research project most of the categories overcame the threshold of saturation after the first ten interviews. At this point, no further categories or properties evolved and all the statements could be assigned to available codes. The last two interviews therefore did not contribute any new information, but increased the density of the existing categories and properties. After the twelve interviews it became easy to integrate the various information and build different *hypotheses* based on it. The result was a set of basic requirements, needs, expectations and wishes of the software development industry. These are outlined in Chapter 5, *Findings of the Interview Series*. Some of the categories did not become saturated. This was either because these categories were not the focus of my studies, and I consequently did no further research on them, or the information concerned particular variations of a known phenomenon or represented a special case. Where such a variation or special case was observed, I noted this in the Findings chapter.

115 In a final step, several theories each relating to a particular problem were elaborated. Hage defines a theory as a "set of well-developed categories (themes, concepts) that are systematically interrelated through statements of relationship to form a theoretical framework that explains some phenomenon".[145] The pattern that is observed during data analysis is used to build generalizable statements. Glaser and Strauss distinguish between two particular types of theories that can be elaborated: The *substantive theory* refers to a particular area of study that was analysed with empirical socio-scientific methods such as patient care, race relations, professional education, etc. The researcher here works on a particular limited frame, focusing on one sociological surrounding in which he or she wants to perceive further incidents and relations. By *formal theory*, they mean a theory that stands for a whole conceptual area of sociological inquiry, such as socialization, authority, deviant behaviour and so on. The researcher looks at the research question on a bigger scale and fo-

143 Corbin/Strauss, 139 f., 141, 188 and 190.

144 Charmaz, 114 f.

145 Hage, 34.

cuses his or her analysis on a sociologically larger field of conceptual topics.[146] This project focuses on a very particular field within information management and intellectual property law. The socio-scientific scope is narrow and does not involve further conceptual areas of sociology. Even if general knowledge about the whole field of software engineering could be gained, it would not be appropriate for sociological modelling of a formal theory. This project therefore aimed to develop substantive theories.

To build a substantive theory, the findings in the form of hypotheses from stage three were further processed. Through discussion and further classification, the issues were evaluated from a legal perspective in order to develop potential theories, including how some problems could be solved, or to indicate where further regulatory measures were required. Where necessary, I discuss literature, cases and studies that refer to the same topic. The results are shown in Chapter 6, *Discussion of Selected Problems*. One example of theory building was the use of the categories 'life cycle of software' and one regarding minimal expectation from the 'term of protection'. Each category was used to develop a hypothesis. These two hypotheses were then discussed from a legal perspective. The result of this evaluation was compared with the current terms of protection in both patent law and copyright. The different hypotheses could thus be related to an actual problem, and served to provide regulatory and practical suggestions. 116

After the writing of this thesis had been completed, specialists from various disciplines examined the different hypotheses and findings by means of *member checking* (*Kommunikative Validierung*).[147] The saturated individual results and hypotheses were shown to one software developer and one attorney specializing in ICT and software copyright, for extensive review. They were also shown to several people for discussion in order to evaluate and validate the quality of the data and express how well it fit their understanding of the problem under investigation. They confirmed that the findings resonated with the experience of professionals for whom the research was intended, and that the concepts would be feasible. The results were thus substantiated and could be utilized. 117

This chapter has explained how and why I decided to work with Corbin and Strauss's grounded theory method to analyse and evaluate the data obtained 118

146 For both expressions, see Glaser/Strauss, 32 ff.

147 For more information on member validation, see for example Corbin/Strauss, 198 f.; Steinke, 319 ff.; Lincoln/Guba, 357 ff.; Seale, 468 ff.

from the expert interviews; the systematic and logically structured approach provided me, as an inexperienced researcher in the field of socio-scientific methods, with additional security and a procedure to follow. The statements within the interview transcripts were assigned to particular categories, corresponding to thematic sets, or descriptive properties. After a total of 12 interviews, saturation of the codes was observed. The categories could then be interrelated through integration to form a substantive theory of how software developers work and commercialize, and how this process could be depicted in a legal framework.

Chapter 3: Technical Foundation

This chapter summarizes the results of an extensive literature review in the disciplinary fields of software engineering and project management to give more background and the technical and economic foundation for an understanding of the subsequent findings and the discussion of the interviews. The chapter starts by defining the relevant technical terms and identifying the main characteristics of computer programs. It then outlines how the development process is commonly structured. In order to outline the particular dynamics of the softtware industry, I provide a short overview of the basic economics of software projects. 119

I. Definition of Relevant Terms

This section gives a brief overview of the relevant terminology in software engineering and electronic data transmissions. Only the most important and relevant terms for the subsequent chapters are discussed and usually only one definition is referred to for each term in order to keep it as simple and comprehensible as possible. If no further comments are added, it can be assumed that I agree with the definition and subsequently use the term in an appropriate understanding. Where necessary, further notes are added. 120

All subsequent terms revolve around the topic of *information and communication technology* (*ICT*) which is understood as every process that involves accessing, storing, manipulation or transmission of electronic data and information in a digital form.[148] 121

[148] Foldoc Dictionary, "information and communication technology", available at <http://foldoc.org/Information and Communication Technology> (retrieved September 6, 2021).

A. The Science of Software Engineering

122 The Merriam-Webster Dictionary defines *software engineering* as...

> "... a branch of computer science that deals with the design, implementation, and maintenance of complex computer programs".[149]

123 *Software* in this context refers to an entire set of computer programs and processes as well as the related documentation associated with a computer system.[150] The process that is necessary to obtain a software product is referred to as *engineering*. It describes the skilful combination and application of knowledge and know-how in science and mathematics to solve a particular problem.[151] It is often also referred to as coding. The term software engineering therefore refers to a systematic study in engineering where mathematics and science are applied to form computer programs which can then solve problems and function in a particular way.

124 I refer to *computer programs* or simply *programs* as self-contained, reasonable and explicit syntactic sequences or units of commands that make a computer solve a particular problem by processing the provided data in a specific way and displaying it in a specific form.[152] They can be categorized according to their volume, the number of transactions and functionality, from a temporal aspect, by accuracy and by evaluating how secure they are.[153] The same program can often solve more than one type of problem.[154] While they are frequently used to organize the communication between users, the most common field of application is the analysis, evaluation, adaption and elaboration of

---

149 Merriam-Webster Dictionary, "software engineering", available at <http://www.merriam-webster.com/dictionary/software-engineering> (retrieved September 6, 2021); see a similar definition in ISO/IEC/IEEE 24765:2017-3.3810, with reference to ISO/IEC TR 19759:2016.

150 ISO/IEC/IEEE 24765:2017-3.3783, with reference to IEEE 828-2012, 2.1; Merriam-Webster Dictionary, "software", available at <http://www.merriam-webster.com/dictionary/software> (retrieved September 6, 2021); Becker et al., 52.

151 Singer, 15 f.; ISO/IEC/IEEE 24765:2017-3.1393, with reference to ISO/IEC 2382:2015; Merriam-Webster Dictionary, "engineering", available at <http://www.merriam-webster.com/dictionary/engineering> (retrieved September 6, 2021).

152 See definitions in Model Provisions on the Protection of Computer Software; 17 U.S. Code § 101; ISO/IEC/IEEE 24765:2017-3.726, with reference to ISO/IEC 2382:2015; Rauber (1988), 15.

153 Koreimann, 104 ff.

154 Hommel et al., 28.

concepts to solve particular problems.[155] To date, no common legal definition of the term is available. The legislators of most countries have abdicated responsibility for defining the term so they can keep an open door for future technical developments.[156] In the literature and in practice, the term software is partly used as a synonym for computer programs. Some authors, for example Rauber, differentiate and refer to computer programs as solely the code lines, while using the term software to refer to all the results of a working process, including the comments and protocols.[157] The market's expectations of what should be contained in computer programs have increased over the last decade, as have the expectations of the job of a programmer. Nowadays, it has become natural to add comments and elaborate a protocol or a concept for a final product. "Simple" computer programs consequently have become much more rare. For this reason, I decided to use the term software as a synonym for computer program in this thesis, while aware that this is not indisputable.

Software is usually, at least to some degree, connected to hardware. *Hardware* 125
is a collective term for the physical parts of the computer system that are tangible. It comprises input and output devices (e.g. keyboard, mouse, screen), storage media (e.g. drive) and devices for data transmission (e.g. cables, wires and modem).[158]

The term *computer* is used broadly in this thesis to refer to all electronic de- 126
vices that perform logic operations through processing data and following logical calculations on a digital basis.[159] *Physical machines*, on the other hand, are built of physical components and mechanical governors, such as wires and ca-

155 Koreimann, 18 ff. and 21 ff.; Singer says that a computer program has to be effective in fulfilling all expectations and nothing more. However, it has to work efficiently in terms of time, money and capacity. This outlines how software is used in practice and what purpose it serves (see Singer, 10).

156 See, for example, comments in the 2nd Law Amendment Act to the German UrhG (2. UrhÄndG), BT-Drucks, 12/4022, 9.

157 Rauber (1988), 19.

158 Becker et al., 31; Marly, N 2.

159 From the Cambridge Dictionary, computer, available at <http://dictionary.cambridge.org/dictionary/english/computer> (retrieved September 6, 2021); ISO/IEC/IEEE 24765:2017-3.717, with reference to ISO/IEC/IEEE 24765:2017-3.717, with reference to ISO/IEC 2382:2015; Rauber (1988), 9; Bauknecht/Zehnder, 232.

bles, screws and gears.[160] The term *electronic constructions* is used to refer to devices that apply low voltage currents and use solid circuits to transmit or process analogue data.[161]

127 software product operates within a particular *network environment*, a so-called *information system*. A program or machine represents an element of a larger functional unit that interacts.[162] The context and setting of a software product influences the composition of a system.[163] As the computer program has to run on a specific machine and is expected to conduct certain processes on it, the program is conceived as a small ecosystem of multiple elements that communicate with each other and follow a certain order to fulfil a particular goal.[164] The various elements in a system consequently have to be synchronized and coordinated to address these dependencies, and new elements have to be *compatible* with the rest of the software environment: The mathematical rule in a command, the programming language and the software's digital or hardware environment have to be carefully aligned with each other.[165]

128 A *server* is a computer program or a central computer that provides information and functionalities for other programs or devices, called *clients*. The term 'server' can be used to refer to the physical machine a server program runs on. But it is mainly used to describe a specific network architecture, the *client–server model*, in which, upon request from one or multiple clients, an operation is distributed across and run on either a single device or within a network on several devices simultaneously. Servers can offer various different

160 Samuelson et al., 2321.

161 The Business Dictionary, "electronic", available at <http://www.businessdictionary.com/definition/electronic.html> (retrieved July 27, 2019); Bauknecht/Zehnder, 232.

162 Straub (2011), N 23; Sommerville, 551 ff.

163 ISO/IEC/IEEE 24765:2017-3.1421; ISO/IEC TS 24748-1:2016, 2.20; ISO/IEC/IEEE 15288:2015, 4.1.19; ISO/IEC/IEEE 42010:2011, 3.8; Koreimann, IX and 5 f.; Bauknecht/Zehnder, 82 f. and 121 ff.

164 Koreimann, 7 f.; Sommerville, 286 ff.

165 Some technical decisions, such as selecting a programming language, can also be predetermined by the program's network environment: For example, on "Android" or "iOS" smartphones only very few programming languages can be processed. Working with an entirely different programming language would therefore not be feasible as the device that has to run the software will not be compatible and will not be able to understand the commands. See also discussion in N 396; Hommel et al., 40; Maynard, 41 f.

functionalities, or *services*, from performing specific tasks (e.g. application servers, mail servers, web servers, proxy servers) to providing certain computations and information within a network (e.g. database servers, file servers).[166]

Programming refers to the effort of solving a problem with the systematic use of given information and a computer by formulating, modifying and testing instructions that a computer understands and is able to follow.[167] 129

The instructions in a computer program have to be written in a *programming language* so that the computer comprehends what it has to do and how. There is differentiation here between machine-orientated and high-level programming languages. Machine-orientated languages are usually adapted to the hardware to which they are connected and allow easy transmission into the machine code form. The required instructions can be formulated precisely and at the same time processed rapidly by the computer.[168] The most common examples are Java, C, C and Basic. As the instructions have to be very detailed, for the software engineer, coding with this type of language can be quite tricky.[169] They thus often prefer the high-level languages, which offer their own set of rules and sequences, enabling a more consistent connection of individual commands. It makes programming simpler as it focuses on the engineering problem rather than on the hardware architecture.[170] However, the transmission into machine code becomes more difficult. Common examples of high-level languages are COBOL, Fortran and Visual Basics. 130

A *user* is a person who utilizes a computer program.[171] The term itself does not provide any information about whether the user is allowed to make use of an object from a legal perspective. It also gives no further information about the technical expertise of the person using the program. 131

166 See ISO/IEC/IEEE 24765:2017-3.3705; see also Wikipedia, "server", available at <https://en.wikipedia.org/wiki/Server_(computing)#cite_note-1> (retrieved September 6, 2021), with further references; Sommerville, 499 f.

167 See Hommel et al., 27 f.; ISO/IEC/IEEE 24765:2017-3.3146, with reference to ISO/IEC 2382:2015.

168 Explained in Straub (2001a), 808; see also Gmehlich/Rust, 9; ISO/IEC/IEEE 24765:2017-3.2297.

169 U.S. Congress (1992), 18.

170 Techopedia, "high-level language", available at <https://www.techopedia.com/definition/3925/high-level-language-hll> (retrieved September 6, 2021); see also Straub (2001a), 808; ISO/IEC/IEEE 24765:2017-3.1809.

171 ISO/IEC/IEEE 24765:2017-3.4469, with reference to ISO/IEC 25010:2011, 4.3.16, ISO/IEC TS 24748-1:2016, 2.60, and ISO/IEC/IEEE 15288:2015, 4.1.52.

132 A *software engineer* or *developer* is a person who employs the principles of computer science to design, develop and maintain computer programs and software environments or makes use of them in further development.[172]

133 A software developer may either design their own technical solutions or utilize existing elements of standard software, models or processes, called *tools and program libraries*. In the engineering process, the developer relies on components or works that have been constructed for another project. The set of tools works in a way like a letter case.[173] General-purpose tools and older blocks can be partially or entirely recycled and integrated into the new program. The main effort within the process is put on programming the gateways between the reused components and the surrounding program structure properly so that the blocks work together reliably.[174] Similarly, in *computer-assisted software engineering* the developer may use generators or other devices that create smaller program components to fulfil particular standard tasks, such as screen masks, database designs, etc. A variety of pre-existing blocks, tools and components are available in programming language libraries, on specialized platforms and in online forums.[175] The use of such foreign components reduces the time and energy the engineer has to invest into reinventing common or particular middleware services and their assessment.[176] As the engineer does not need to construct a problem-solving command manually from scratch, his or her work is facilitated and resources are preserved.[177] At the same time a minimum standard can be maintained, encouraging standardization.[178]

134 The Systems Development Life Cycle of software, or for short, the *software life cycle*, refers to the lifespan of a computer program from its development to the moment when it has to be replaced with a new system. A software product's life cycle includes every step in the program's development, its implementa-

172 See ISO/IEC/IEEE 24765:2017-3.1174, ISO/IEC 25000:2014, 4.6, ISO/IEC 25040:2011, 4.12; Wikipedia, "software engineer", available at <https://en.wikipedia.org/wiki/Software_engineer> (retrieved September 6, 2021).

173 Examples of computer-assisted systems are front-end tools which support the engineer with an application-related design of a system, and so-called back-end tools, which use data processing in order to transmit a problem-designed solution into a real software system (for both, see Koreimann, 144 ff.; Bullinger/Faehnrich/Ilg, 954 f.; see also ISO/IEC/IEEE 24765:2017-3.4330, with reference to IEEE 1175.2-2006, 3.16).

174 Koreimann, 168; Sommerville, 57; Bauknecht/Zehnder, 85.

175 Koreimann, 144 ff. and 169; Sommerville, 53 and 440 ff.

176 Schmidt, 26; See Boehm (1981), 678 f.

177 Schmidt, 25.

178 Kuhn; Schmidt, 36 ff.

tion, testing, reviewing and maintenance.[179] The software system represents a dynamic object with behaviour that is flexible to changes and improvements. Over a product's lifespan, it undergoes several maintaining interventions and is enhanced multiple times. It is continually changed in order to adapt it to the new requirements of its environment.[180] Over time, the system keeps growing in complexity, density and quality. With every further change, the software's structure inevitably degenerates. At some point, the software system cannot be amended or maintained in a way where it is still able to function effectively and in an economic manner, which is when it has to be disposed of and replaced with a new system.[181] If the project to be developed exhibits a long life cycle,[182] it is reasonable to consider a plan for longevity during the design and development of the computer program.[183] Through smaller details, such as carefully selecting an appropriate development approach, properly elaborating the structure and formulation of the source code, and building reliable connections between the different modules, a longer life cycle of the software is possible.[184]

In software engineering, there is often differentiation between analogue and digital procedures. These are the different ways a signal is transmitted and the different channels that are used to do so. In *analogue* transmissions, as they are used for terrestrial TV and tape, constantly changing electrical voltage is continuously passed along the system. In *digital* transmissions, on the other 135

179 Foldoc Dictionary, "systems development life cycle", available at <https://foldoc.org/Systems Development Life Cycle> (retrieved September 6, 2021); ISO/IEC/IEEE 24765:2017-3.2217, with reference to ISO/IEC 12207:2017, 4.16, ISO/IEC TS 24748-1:2016, 2.24, and ISO/IEC/IEEE 15288:2015, 4.1.23.

180 See for example Swanson/Dans, 278; Belady/Lehman, 227; Barry/Kemerer/Slaughter, 206; Wirth, 72 f.

181 According to the second law of program evolution dynamics, the law of increasing entropy, the unstructured nature of a system increases with time, unless specific work is executed to maintain or reduce it (see Belady/Lehman, 228); see also Swanson/Dans, 279 f. and 285.

182 For more information on the software life cycle, see N 134.

183 Barry/Kemerer/Slaughter, 206; referring to Swanson/Dans.

184 Boehm (1981), 29.

hand, the electrical voltage is quantized and staggered. Mostly used for data storage on compact discs or computers, the electrical voltage is expressed in numerical values (0 and 1s), or digits.[185]

136 *Reverse engineering* is used when someone tries to gain understanding and extract knowledge about a process or good to find out how it is constructed.[186] The structures, condition and behaviour of a software product are observed and dissected in order to identify the technical components and construction elements involved. In the context of intellectual property law, reverse engineering is often associated with the fact that competitors could misuse this approach to explore an invention.[187]

137 *Decompiling*, on the other hand, is when the machine code of a computer program is converted into a human readable version, such as a source code.[188] The conversion usually has the same functionality as the original source code, but not necessarily the same literary formulation.

B. Elements of a Computer Program

138 *Data* are words, numbers and characters that carry and mark specific information such as facts, concepts or instructions in a form suitable for communication, interpretation or processing.[189] The raw data work as an input and have to be processed through the software.[190,191]

---

185 There is rarely a comprehensible or citable definition available for the terms 'digital' and 'analogue'. For a good example, see: Wikipedia, "analogue electronics", available at <https://en.wikipedia.org/wiki/Analogue_electronics> (retrieved September 6, 2021); Wikipedia, "digital electronics", available at <https://en.wikipedia.org/wiki/Digital_electronics> (retrieved September 6, 2021). BORNHAUSER, N 14 ff.

186 Merriam-Webster Dictionary, "reverse engineering", available at <http://www.merriam-webster.com/dictionary/reverse engineer> (retrieved September 6, 2021); ISO/IEC/IEEE 24765:2017-3.3501.

187 SAMUELSON ET AL., 2401 ff. Due to its scope, reverse engineering is not the subject of this thesis. However, for more information on reverse engineering in copyright, see SAMUELSON/SCOTCHMER and RAUBER (1992), 39 ff.

188 Techopedia, "decompile", available at <https://www.techopedia.com/definition/16374/-decompile> (retrieved September 6, 2021); ISO/IEC/IEEE 24765:2017-3.1076.

189 See ISO/IEC/IEEE 24765:2017-3.985; STRAUB (2011), N 241 ff.; RAUBER (1988), 4, with further references.

190 HARISON, 173.

191 It is important to distinguish between data as input and the software processing the data.

Several data combined, which carry information to the same issue form a *dataset*.[192] 139

A *database* represents a comprehensive collection of related process data.[193] Single records of data, whole sets or even files may be retrieved or grouped according to chosen criteria or specifications.[194] 140

A *bit* (short for binary digit) refers to the smallest identifiable unit in which information can be packed. Bits are displayed with the digits 0 (*zero*) and 1 (*one*) and when joined together carry electronic commands in the form of single stimuli that can be processed by machines.[195] Eight bits are grouped together and function as one addressable memory unit,[196] a *byte*. The representation of an electronic command in bits and bytes is called *binary*, *object* or *machine code*. 141

Programs usually consist of written instructions.[197] These can be represented in two different forms: *source code*, which is written by a computer engineer in a specific programming language and put into a logic structure suitable to be translated into a computer-understandable form, and *machine code* which is formulated in binary code and is recognizable by a machine in order to carry out the programs.[198] Machine code and source code therefore are representations of the same command but displayed differently for the machine and the programmer. In contrast to machine code, source code may be displayed directly on the screen of a computer and can be copied, deleted and edited on screen by hand. It can also be printed on paper.[199] A *compiler* or *assembler* then translates the source code into machine code so that the computer can process the changes in instructions. The compiler itself is a computer program 142

192 Bauknecht/Zehnder, 24 f.; Rauber (1988), 4.

193 Dictionary.com, "database", available at <https://www.dictionary.com/browse/database> (retrieved September 6, 2021).

194 Rauber (1988), 5, with further references.

195 Techterms, "bit", available at <http://techterms.com/definition/bit> (retrieved September 6, 2021); ISO/IEC/IEEE 24765:2017-3.388, with reference to ISO/IEC 2382:2015, 3.2.

196 Rauber (1988), 8 f., with reference to Bauknecht/Zehnder, 22 ff.; ISO/IEC/IEEE 24765:2017-3.450, 3.3.

197 Harison, 173. Programs can also consist of purely visual commands, for example in the form of flow charts, but this is the exception.

198 Koglin, 7 f.; Bauknecht/Zehnder, 83 f.; U.S. Congress (1992), 7; ISO/IEC/IEEE 24765:2017-3.3882 and ISO/IEC/IEEE 24765:2017-3.2296.

199 See also Boecker, 46 f.

as well.[200] It either translates the whole program at once or focuses on individual instructions for converting.[201] The source code usually specifies which data the program wants to process and how it wants to do so.[202] While this sensitive information is visible in the source code, the machine language form has the advantage that it is not 'readable' for humans as it simply shows the figures 0 and 1. Consequently, one cannot recognize sensitive information directly.[203] Software developers often prefer to release their software in machine code form instead of providing the decipherable source code in order to protect the know-how contained in the program.[204]

143 Software engineers may also add *comments* to the source code. Comments are "normal" text lines that are placed in the source code and explain the processes and variables included in the code or simply provide further information, for example to co-authors or other engineers.[205] The comments are created for humans only and do not affect the machine interpretation of the code.[206]

144 *Documentation* is supplied with a computer program and contains written or pictorial information describing and specifying the software product in terms of included requirements, procedures and results. It also includes instructions that explain to the user how to utilize and maintain a product.[207] The documentation usually consists of separate notes, a specification sheet, concepts or sketches. In a later stage of production, a documentary manual may also be delivered with the final software product.[208] Separate from the regular documentation that is handed to the users are documents and sketches that were produced mainly during the conceptualization phase and cover all the notes that refer to the development process.[209] In this thesis, I refer to this kind of documentation as *development documentation*.

200 ISO/IEC/IEEE 24765:2017-3.681; Hommel et al., 30; Bauknecht/Zehnder, 83 f.

201 Perelman, 923 f.

202 Haberstumpf (1993), II N 17.

203 See also U.S. Congress (1992), 7.

204 Perelman, 923.

205 Koglin, 7 f.; Hommel et al., 151; ISO/IEC/IEEE 24765:2017-3.641.

206 ISO/IEC/IEEE 24765:2017-3.641.

207 ISO/IEC/IEEE 24765:2017-3.1259; Becker et al., 223 f.; Marly, N 12 f.; Thomann (1992), 2.

208 Singer, 135 ff.

209 Bauknecht/Zehnder, 272; Becker et al., 200, 209 and 225; Thomann (1992), 2.

An *algorithm* is a mathematical object that expresses one or a set of well-defined rules to solve a particular problem in a finite number of steps.[210] In the field of computer programs, the commands consist of rules that define the sequence of arranged steps in order to carry out a particular function or generate a specific behaviour in a program.[211] Information is processed strictly according to the rules, resulting in a specific output, e.g. data.[212] If the algorithm is repeated with the same variables, the same result will always be obtained.[213] There is a distinction between basic algorithms and complex algorithms.[214] *Basic algorithms* consist of basic modules that are frequently used in software engineering, suitable for all kinds of plain problems.[215] *Complex algorithms*, on the other hand, include a first minimal structure of a potential solution to a particular problem, usually combining basic algorithms and less complex algorithms to concretize tangible teaching.[216] 145

The *external design* or *graphical user interface* (GUI) is an interactive component used as a communication medium between the program and the user. It is the visual image of the computer program, which facilitates the use and handling of it on a computer. Instead of working with a technical source or machine code, the user steers and controls the software with graphical control elements, such as symbols, buttons and dialogue windows. The graphical input is then translated into impulses the machine understands.[217] GUIs make the use of the program more attractive and increase its usability. The external design is conceptually separable from the program code that transcribes the interface. 146

The *look-and-feel* refers to the way the user interface is perceived and can be operated technically.[218] The component 'look' refers to the visual appearance of a user interface, especially its layout in the form of graphics, fonts and 147

---

210 See ISO/IEC/IEEE 24765:2017-3.124, with reference to ISO/IEC 2382:2015; Samuelson et al., 2384; Hommel et al., 29; Gmehlich/Rust, 9 ff.

211 Harison, 173; see also Calame (2007), 328 f.; Zirn, 9.

212 Hommel et al., 29; see also Haberstumpf (1983), 240 ff., for further differentiation.

213 See Rauber (1988), 20 f.

214 See detailed distinction in N 700; see suggested model in Ensthaler/Moellenkamp, 152 ff. They appear to have based their subdivision partly on Sommerville, 4th edition of 1992, see particularly vi, 133 f., 169 ff. and 590 ff.

215 Ensthaler/Moellenkamp, 152 f.; see also Floyd, 459; Straub (2011), N 18.

216 Ernsthaler/Moellenkamp, 153 f.; see also Floyd, 459;

217 U.S. Congress (1992), 18; ISO/IEC/IEEE 24765:2017-3.4475, with reference to ISO/IEC TR 25060:2010, 2.23; Straub (2011), N 18; Bullinger/Faehnrich/Ilg, 941; Schlatter, III N 68.

218 Schwabach, 167.

colours, etc. The term 'feel' refers to the user's experience of using a computer program, the visually perceptible features of the software which are integrated into the layout. This may, for example, include how the user navigates through the software with buttons, menus, links and dialogue boxes.[219] It is often related to a particular feeling that suits the characteristics of the software vendors, the respective computer program, or both. A more elaborate or specified look-and-feel often refers to a higher usability of the program for the final user. It thus mainly serves a business function and aims to make the computer program more attractive for the user.[220]

148 Another important structural component of a computer program is the functions and features. The *features* of a computer program are the characteristics it offers to fulfil a specific need for a user.[221] *Functions*, on the other hand, refer to the technical and visual activities, performance or capabilities a program offers and its requirements.[222] They are the specific actions that the 'software will fulfil', built to perform particular tasks a user requires[223] To take the example of a car, the navigation system represents a common feature of the car whose function is to guide the driver where to go. In the case of a computer program, a typical feature of a web page, for example, is RSA-encryption.[224] Its function is to secure data transmission. All the functions of a computer program are usually described in the functional specification document of a software product or predefined in a requirement sheet.

149 The interaction between different software and hardware components happens through *interfaces*. They work as connecting points between different units, passing information from one to the other, ensuring data flow. For this purpose, they have to define the characteristics of which information is allowed to access and pass through to the next connecting unit, to ensure that each unit is able to understand the data delivered and then process it correctly.[225]

---

219 Schwabach, 167.

220 Schlatter, III N 69 f.

221 ISO/IEC/IEEE 24765:2017-3.3814 and ISO/IEC/IEEE 24765:2017-3.1582, with reference to ISO/IEC/IEEE 23026:2015, 4.9, and ISO/IEC/IEEE 26515:2011, 4.6.

222 ISO/IEC/IEEE 24765:2017-3.3814 and ISO/IEC/IEEE 24765:2017-3.1677, with reference to ISO/IEC 26514:2008, 4.21.

223 See Bridging the Gap, "functional specification", available at <http://www.bridging-the-gap.com/functional-specification/> (retrieved September 6, 2021).

224 Apel does not provide a definition for the term 'function' but refers to RSA encryption as an example (see Apel, 7).

225 See ISO/IEC/IEEE 24765:2017-3.3.2058; see also Straub (2011), N 24.

C. Typology of Computer Programs

The term software is a generic umbrella term that includes different types and forms of computer programs. Products can be classified into system software and application software, although mixed forms are also available on the market. 150

System software refers to programs designed to facilitate controlling, operating and maintaining a computer system and its associated programs. It also involves operating systems, device drivers and utilities.[226] 151

Application software as a collective term refers to all programs that solve a particular problem of data processing for users, for example text editing, accounting and so on.[227] This term may be further subdivided into the categories of standard software and individual software. *Standard software* refers to a particular kind of software that targets a wide community of users by offering standardized products for off-the-shelf solutions and common industry issues.[228] Commonly they are elementary programs at a mid or lower price range which do not need much support from the program issuer.[229] On the other hand, *individual software* is directed to the specific needs of a particular group of users and tries to solve their problems.[230] As they are customized, they are usually more cost-intensive. 152

The term software is used extensively and includes macros, program libraries[231] and computer games. *Macros* are stored and retrievable command sequences within a program.[232] *Computer games* are video games that are played on a computer rather than on a console.[233] 153

---

226 U.S. Congress (1990), 6; ISO/IEC/IEEE 24765:2017-3.4116; Slongo, 4 f.; Becker et al., 52; Thomann (1998), 11.

227 ISO/IEC/IEEE 24765:2017-3.192, with reference to ISO/IEC 2382:2015; Kindermann, 151; Becker et al., 53 f.; Thomann (1998), 10 f.

228 Kindermann, 151; Bauknecht/Zehnder, 109.

229 Rueesch, 23.

230 Kindermann, 151.

231 For more information on program libraries and tool-assisted software engineering, see above N 133.

232 Straub (2011), N 25 and 51; ISO/IEC/IEEE 24765:2017-3.2301.

233 Techopedia, "PC game", available at <https://www.techopedia.com/definition/31136/personal-computer-game-pc-game> (retrieved September 6, 2021).

154 One can also distinguish between computer programs that are connected to hardware components and are therefore called *hardware-related programs*, and *computer implemented programs*. The latter are programs that solely work within the computer, a network or other programmed devices.[234]

155 Some computer programs are able to perform their functions independently, without requesting a further connection to another component or device. They are called *stand-alone* programs.[235] Others rely on a pre-developed basic program, working merely as additional programs or extensions. Examples of dependent software are add-ons, plug-ins and updates. An *add-on* refers to a computer program that can be added to a pre-existing one in order to increase the number of functionalities it can perform.[236] A *plug-in*, on the other hand, refers to a whole software component that can be added into a previous application to increase the number of features the program includes.[237] *Updates* consist of additional computer programs or files that are used to fix specific problems in earlier versions of a computer program or apply changes that make it better operable.[238] All these supplements are commonly supplied by the originating seller of a software product but sometimes are also provided by other companies, institutions or individual engineers. They either are distributed on a physical medium, such as a CD, or are accessible online.[239]

II. The Development Process

156 The development process in software engineering focuses on how the solution to a particular problem evolves from defining the problem through to formulating an instruction the machine is able to understand. On a temporal axis, it involves every creative and innovative step from a first idea to a final product that can be implemented or offered on the market. The development process has been described by a number of different authors, focusing on various aspects of development. All of them give an abstract description of a process ar-

234 See distinction according to the Guidelines for Examination of the European Patent Office, Part G-II-3.6.

235 ISO/IEC/IEEE 24765:2017-3.3948.

236 Macmillan Dictionary, "add-on", available at <https://www.macmillandictionary.com/dictionary/british/add-on_2> (retrieved September 6, 2021).

237 Wikipedia, "plug-in", available at <https://en.wikipedia.org/wiki/Plug-in_(computing)> (retrieved September 6, 2021).

238 See Computerhope, "update", available at <https://www.computerhope.com/jargon/u/update.htm> (retrieved September 6, 2021).

239 See also discussion in Hilty (1997), 128 and 135 f.; Lemley et al., 31.

chitecture, design or definition.[240] They all agree that a customized methodological approach has to be chosen for each engineering project. The following chapter tries to systemize the complex process of software development into basic phases to give a rough idea of what models are available to structure a development project.

Those developing a software product may vary between independent individuals, small or large companies, public corporations and academic institutions. In the development process, software engineers but also product managers and other departments are often consulted. Sometimes, even third-party companies are involved, especially in the case of commissioned work. All these different people may be associated with the development process. The term 'software developer' is therefore used quite broadly in this thesis and may refer to humans as well as entities that are involved in developing a software product. 157

A. The Standard Phase Model

In computer science, various models have been formulated to describe the different stages that appear during software development. Some models capture the process as chronologically ordered different phases and are, therefore, called *phase models*. Others use a set of standards for systems analysis and application design and focus on illustrating and clustering the relationship between the input and output of assigned tasks. This approach is called structured systems analysis and design methodology.[241] For the present study, I decided to build my research on the phase model in order to understand the constructive course of the development process rather than just the technical structures behind it. In the following, I will therefore take a closer look at the possible sequences of such a phase model. 158

The literature does not offer a universal division of the development process into specific phases, nor does it work with a consistent nomenclature.[242] How- 159

240 WEN/TUFFLEY/ROUT, 3.

241 Common examples of structured system analysis methods are: systematic structure models, the Yourdon Structured Method, SADT diagrams, the Problem Statement Laguage and Petri net simulations. For a detailed description see: KOREIMANN, 153-160; see also description in: Techopedia, "Structured Systems Analysis And Design Method ", available at <https://www.techopedia.com/definition/3983/structured-systems-analysis-and-design-method-ssadm> (retrieved September 6, 2021).

242 Same conclusion in: RAUBER (1988), 21.

ever, common features and processes can be observed that together form what I will define as the *Standard Phase Model* in software development. Each phase can be associated with particular decisions and procedures the developer has to follow to produce his or her computer program.

160 As Singer described in his book in 1984, everything starts with an idea: "The human sits at a table, prepares a blank sheet of paper, takes a pencil and thinks."[243] Singer explains that when an idea develops, the engineer does not directly write down code lines. Instead, the inventor takes notes of their idea and develops what they want to achieve with their program, and which functions it should possess. These complexes can then be used to break down the ideas into detailed drafts. A verbal formulation, e.g. in a product specification or duty book, is produced to provide some precision for the basic idea. When all potential questions have been answered, the coding can start.[244] Although Singer's depiction of working with a pencil and a piece of paper today would be considered somewhat outdated, it functions as a nostalgic illustration. What we learn from it is that software development commonly consists of three main stages: an idea, a drafting process in which the developer reflects on what he or she wants to achieve and what functions the software needs to process, and a final coding phase. Similarly, Rauber distinguishes between the analysis phase, the drafting phase, and the final coding.[245] In his description, Koreimann distinguishes between preliminary studies and system analysis, but also mentions a realization phase, in which coding is done. He further supplements the phase model with the implementation of the software in its future environment.[246] Today, the product's implementation is commonly combined with testing and reviewing so it can be smoothly integrated into its new surroundings and the issuer can ensure that it functions, and the model is regularly complemented with maintainance as a last phase. To safeguard the program's functions, the issuers carry out maintenance once it is implemented and running.[247]

161 For this study, I decided to work with my own phase model, which I aligned broadly with Singer's but with further elements to give a fuller understanding of the process. Most of the development work, software development in its *narrower sense*, happens in what I refer to as ideation, conceptualization and

243 Singer, 9.

244 For a full illustration with further explanation see: Singer, 9 ff.

245 Rauber (1988), 21 f.

246 Rauber (1988), 21 f.

247 Boehm (1981), 54 f.; Bauknecht/Zehnder, 280 f.; Becker et al., 74.

realization. The implementation and review or maintenance just represent a structural logical consequence of the development process itself, although complementing the concept of software development in a *broader sense*. These five phases, as illustrated in the following figure, together make up what I refer to as the *Standard Phase Model*. Each phase commonly consists of a series of structural steps, and the end of each phase is marked by a particular deliverable that needs to be accomplished, subdivisible into the idea, the concept or draft and the realized software. The deliverable to be achieved usually gives the phase its name, e.g. the "concept" that is elaborated during "conceptualization".

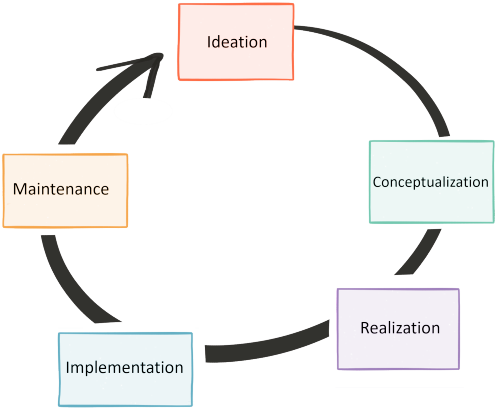


Figure 1. The Standard Phase Model (Source: own illustration).

In this theoretical model of phases, a developer can only continue to the next phase if an intermediary deliverable at the end of the previous phase has been achieved and he or she decides to continue to the next step. The individual phases should therefore be clearly separable and delimitable. Boecker asserts that the systematization of the development process into such distinctive phases is rather used as a theoretical model to simplify the process. In practice, the phases are usually not implemented one-to-one.[248] In respect of the interview series, I would agree with Boecker that there are variations and combinations of the Standard Phase Model in actual software development. However, as will be shown later in this thesis, in practice, the different named phases can indeed be observed in every development process in one form or

162

[248] Boecker, 43.

another. Due to practical circumstances, certain elements of phases may partially overlap, particularly in agile development, but the basic structure of the individual phases can still be delimited and separated. The proposed model thus realistically reflects the relevant software development processes, while at the same time illustrating the complex technical procedure to a lay person. The five phases of the Standard Phase Model can be set out as follow:

1. Ideation

163 The occurrence of an idea represents the starting point of all projects. It can be triggered by any kind of incident, experience or observation. While this may sound like a passive process that just 'occurs', ideation is usually actively sought by individual developers or whole business or product teams, e.g. through design thinking.[249] It involves reflecting on how a particular problem could be approached and what is necessary to solve it. The developer, development team or project manager analyses potential organizational problems and evaluates them. They design a first conceptual framework and try to anticipate which monetary, informational and functional characteristics the outcome should feature.[250] This helps to assess how feasible the project is.[251]

164 Existing approaches of competitors on the market are examined and compared to the approach to be developed. The developer also outlines what potential needs and expectations a customer could have, to understand which services should be implemented.[252] The developer defines the requirements that a potential draft should fulfil and whether, from this perspective, the project and the product can be realized reasonably from an economic and technical standpoint.[253] At the end of ideation, the project manager will have visualized their idea and what it might be able to achieve. Before the next phase starts, the project manager has to decide whether the project should be advanced or whether it should be terminated.

---

249 Design thinking refers to a specific cognitive and strategic process which aims to lead to the design of ideas and proposals for new products and services. For more information, see: Cross, 79 ff.; The Business Dictionary, "design thinking", available at <http://www.businessdictionary.com/definition/design-thinking.html> (retrieved July 27, 2019).

250 Koreimann, 154; Zehnder, 29 f.

251 Slongo, 9 f.

252 Edvardsson, 40.

253 Koreimann, 155 f.; Edvardsson refers here to an exhaustive market analysis, where the offers of competitors and the needs of potential clients are evaluated (Edvardsson, 40 ff.).

2. Conceptualization

If the development team, or their supervisors, decide to continue with the project, the rough notes from ideation can be transformed into a formulated concept, including a specification of requirements, first drafts, sketches and further suggestions as to how the idea can be realized. It involves several steps, from the preliminary concept, through extensive system analysis, to the design of the system.[254] During these steps, the developer has to consider and integrate technical factors, client needs, economic viability, applicability, legal and other criteria in order to obtain a prudent concept. While technical aspects, such as compatibility and capacity, are particularly important for the later conception of the algorithm as well as the source and machine code,[255] the other factors represent relevant business considerations that software developers include in their decision-making in order to address a program's usability and define its merit. 165

Already at this early stage, the developer or engineer will try to draft a rough conceptual formulation of a potential program. They will set the concept in the context of their personal story; the project manager has to describe to what extent their project offers added value over past approaches. They will use numbers and figures to illustrate their plan and provide a reasonable basis for future decision-making.[256] With the help of *system design* the engineer drafts the individual elements of a software product, including its architecture, modules, interfaces and data inputs and outputs. They then align all elements with the surrounding environment or network.[257] The engineer therefore focuses on the software content and all the organizational decisions for the requirements of the software product, developing a coherent and well-run system.[258] The developer decides on the optimal programming language to use and what technical requirements have to be met to facilitate the later implementation of 166

254 Bauknecht/Zehnder, 86, 94 f. and 99.

255 Harison, 176.

256 See Koreimann, 156 f.; Edvardsson, 39 f.

257 See short explanation in: The Medium; see also: The Business Dictionary, "system design", available at <http://www.businessdictionary.com/definition/system-design.html> (retrieved July 27, 2019); Techopedia, "system design", available at <https://www.techopedia.com/definition/29998/system-design> (retrieved July 27, 2019).

258 Techopedia, "system design", available at <https://www.techopedia.com/definition/29998/system-design> (retrieved September 6, 2021); The Business Dictionary, "system design", available at <http://www.businessdictionary.com/definition/system-design.html> (retrieved July 27, 2019).

the program into its future operating environment. The concept should feature a description of the components, including a written comparison of the different engineering methods that could be used, as well as the minimum system specifications.[259] One could even provide the first draft of an algorithm, if available and necessary. At the end of conceptualization, the developer has an advanced draft with detailed information, which serves as a base to decide whether the solution is applicable for the particular problem and whether the project should be realized.[260] It is summarized in the development documentation.[261]

3. Realization

167 During realization the engineer develops the final program based on the detailed concept of the previous phase, refining, reviewing and supplementing it.[262] In this phase, the actual engineering starts, where the engineer builds the desired commands through coding. The developer has to show high constructivist thinking and the capacity to express the verbal instructions in precise commands that the computer can understand. The problem to be solved – the desired behaviour of the computer, or simply how it should act – is illustrated in a rule which determines the repetitions, sequences and selection activities the computer program will carry out.[263] The program is individualized through its mathematical structure by integrating particular variables in parameters and different factors into the final algorithm.[264] In most cases, a single mathematical rule can be represented in many different forms, including two different organizational levels (*source code versus machine code*), various programming languages and, within the code, in different structural places (*sequences*, *structure organization* etc.).[265] An engineer or a graphic designer implements the user interface and the 'look-and-feel' of the program by transcribing it. They may also make use of and integrate available tools or work with computer-assisted methods.[266] Realization is completed when the program has

259 SLONGO, 11.

260 RAUBER (1988), 20 and 22; STRAUB (2011), N 6; ZEHNDER, 64 f.

261 See above N 144.

262 ZEHNDER, 86.

263 HOMMEL ET AL., 41-45; BECKER ET AL., 72 and 219 f.

264 See also HOMMEL ET AL., 37.

265 MOEHRING, 273 f.; U.S. CONGRESS (1992), 17; ULMER, 17 f.; see also discussion in BOECKER, 147.

266 BAUKNECHT/ZEHNDER, 106 f.

reached the quality to be delivered to the customer for implementation. Comprehensive *documentation*, including comments within the source code, is prepared and enclosed before it is delivered to the customer.[267]

4. Implementation

Once the computer product is completed, the programmed software is translated into its final hardware or software environment by handing it over to the buyer, customer or IT department.[268] The product is integrated into its new operating environment, installing it on or linking it to the computer that will utilize the applications. In a modern understanding, implementing a computer program usually involves *testing*, either done by the issuer in a specific simulation environment or by the customer after the software has been integrated into its final system environment, to analyse whether the computer program is able to fulfil the intended purpose, what performance can be achieved, and if further review, improvements or maintenance are necessary.[269] At this stage, a user-friendly interface is important, so that people who have no particular technical knowledge in using computer programs can use the product and benefit from it. If a linear development approach is followed,[270] the development process in a narrower sense ends here, and all connections to previous organizational and auditing measures of the software issuer are loosened. At the same time, the issuer may – if contractually eligible – publish further copies of the computer program and establish them on the market through merchandizing or commercialization.[271] In spiral development, on the other hand, testing and reviewing is used to construct the next modules, thus building the starting point for the next cycle.[272] 168

5. Review and Maintenance

After the computer program is delivered to the customer, the software product is reviewed and maintained. The developing company remains responsible for potential errors within the scope of guarantee services and warranties. In this case, the operational software may need to be modified upon request. At 169

267 Bauknecht/Zehnder, 272 f.

268 ISO/IEC/IEEE 24765:2017-3.1891, with reference to ISO/IEC 19500-2:2012, 3.2.8.

269 See Koreimann, 157 f.: Slongo, 13 ff.; Bauknecht/Zehnder, 87; Becker et al., 72 f. and 220 f.

270 See N 172 ff. for more information on the linear development process.

271 See following Chapter 3 Section III. B. for more information.

272 Cockburn, 27 and 28; Larman/Basili, 48 f.; Ewusi-Mensah, 76; Becker et al., 75; see N 175 ff. for more information on the spiral development process.

this stage of the development process (in a broader sense), the primary functions, as previously developed, remain largely unaltered and only single, particularly selected parts of the source code, graphical user interface, documentation or database structure are reviewed and thus corrected, adapted or improved.[273] In spiral development, particular parts of the software are reworked and improved after implementation of a module, while new modules are built in circular repetition.[274]

B. Three Different Approaches for Developing Software

170 The software development process can be structured either from a temporal perspective or on the basis of the specific problem that a developer has to solve. Based on this division, there are currently two main models that are discussed in the literature: the linear approach, which is constructed along the development phases and works through them chronologically, and the spiral approach, which focuses on building the different program features module by module, taking loops between ideation, conceptualization and realization for each module to be constructed. The third approach, called continuous delivery, is a rather new phenomenon that conflicts with the classic phase models. It refers to an approach where the software is run on an online server and remains accessible and adaptable for the issuers, once implemented.

171 As the differences between the models are rather difficult to understand for a lay person, the models are explained with the help of bridge construction, an illustration proposed by Caroli in 2008,[275] and adapted and advanced for this doctoral thesis.

1. Linear Development

172 In 1970, Royce introduced what he called the linear development approach and what was later partially referred to as the *Waterfall Model*.[276] This approach is said to be the first ever and also, today, the most common project development

273 Boehm (1981), 54 f.; Bauknecht/Zehnder, 280 f.; Becker et al., 74; ISO/IEC/IEEE 24765:2017-3.2318, with reference to ISO/IEC 25051:2014, 4.1.9.

274 See N 175 ff. for further references.

275 See Caroli; architecture is a common illustration used to explain software development projects because software construction partially follows the same business, creative and technical ideas that conceptualizing a building does. The concept of a computer program is therefore also referred to as 'software architecture'.

276 See Royce; for a full description, see also Boehm (1981), 35 ff.

approach (not necessarily in software engineering).[277] In this model, computer programs are developed phase-by-phase through the temporary stages, possibly with overlap but with little or no iteration.[278] The developer consequently follows a deterministic and straight-line procedure along the phases of ideation, conceptualization and realization one-by-one, before implementing the product into its final environment. Only there, afterwards, potential maintainance happens, whereby the general structure of the released program stays unaltered and only smaller parts are maintained. As the following graphic shows, the previous phase has to be completed before the developer goes on to the next one. There is only one course of direction. For this, the developer has to verify and validate the previous results before continuing. Like a river, he or she never returns to a previous stage, but instead always moves forward in the project line.[279] As the characteristics of the linear development model widely correspond to the Standard Phase Model, the above can be transferred to it analogously.

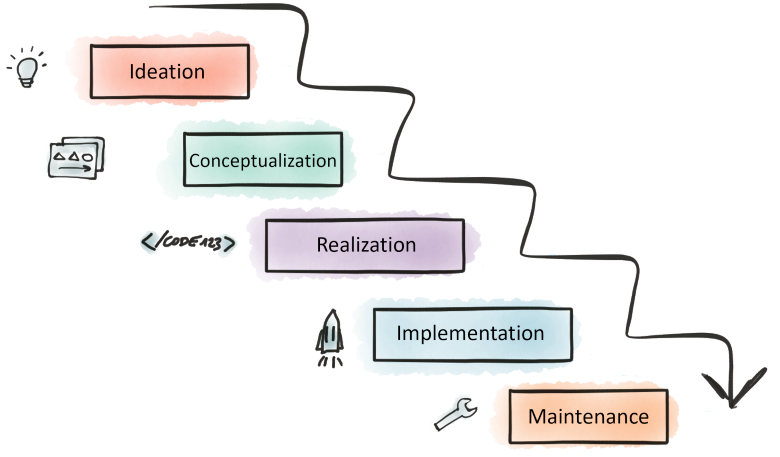


Figure 2. The Linear Development Approach (Source: own illustration).

277 One reason why it is as famous as it is, is because it was adopted by the U.S. Government, making it the standard software development process for military defence systems in 1985 (see U.S. Department for Defense).

278 ISO/IEC/IEEE 24765:2017-3.4584.

279 Bell/Thayer, 62; Maynard, 1 f.; Bock, 241 ff.; Boehm (1981), 36 f.; Becker et al., 72 f. and 75 f.

173 As noted above, each of the development models can be illustrated with the help of bridge construction. The abstract idea of the subsequently illustrated thought experiment (Figure 3) is to build a bridge over a small creek that separates two shores. With the linear approach, the constructor first does some research on which methods are available for building bridges. He or she learns that in classic architecture a bridge is usually built from both sides simultaneously. The constructor takes into account all the relevant requirements, depending on the surroundings of the potential bridge, its base, the material used, the purpose it will serve, the available financial means and so on. The goal is to build a bridge that withstands all kinds of weather and is able to be crossed by one person, a group, a bicycle or a heavier means of transport. Having settled all these organizational matters – knowing the requirements and necessities – the constructor draws some sketches, based on how the problem can be solved, and how he or she wants to build the bridge. They then concretize these sketches until they have obtained a detailed draft and know quite accurately how the plan on the paper has to be realized step-by-step. Then the constructor implements the plan physically and starts putting one brick on another. Only when the mortar is dry is the bridge completed and ready to be used.[280]

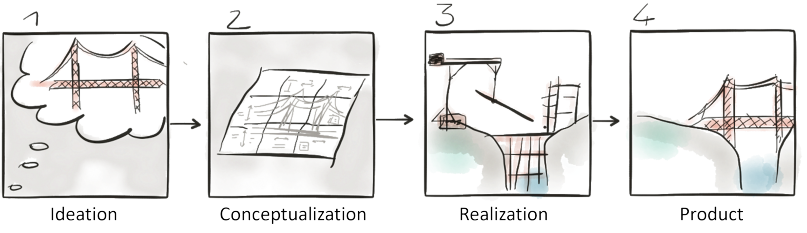


Figure 3. Bridge Construction According to the Linear Development Approach (Source: own illustration).

280 See CAROLI.

The biggest benefit of the linear approach is that the program is usually more compatible with its network environment, as each phase is advanced in close connection with the previous findings and the planning process. The relevant data structures can be pulled through the entire process.[281] It helps the engineer to maintain a clear overview of the whole project and to know exactly which phase they are at and which steps follow next. McConell, in this context, emphasizes that extensive conceptualization and drafting of the requirements can reduce later costs for expensive improvements and corrective measures to coding and maintenance.[282] The linear approach also ensures detailed and close documentation and secures know-how for later utilization.[283] The waterfall model represents a comprehensible model that provides a basic structure understandable to laypersons. 174

2. Spiral Development

In spiral or modular development, the development process is not coordinated linearly along the development phases, but instead focuses on the different modules the software developer wants to build in time cycles.[284] The developer starts with one particular problem he or she wants to solve, specifies the requirements, designs the concept, does the necessary coding for the particular module and connects the elements with the existing parts.[285] We can distinguish between incremental, iterative and agile properties in spiral development – although they are often combined and hard to distinguish in practice: 175

The term *iterative* refers to the planned repetition of certain development phases; when the developer tackles one particular problem, he or she goes through all or at least several phases of the standard phase model – ideation, conceptualization, realization, implementation and review (instead of classic maintainance) – within one loop. However, instead of processing the phases for the whole project sequentially, the developer repeats them spirally for each module, forming loops of phases as illustrated in Figure 4. Through this repe- 176

281 See also Ensthaler/Moellenkamp, 156.

282 McConell, 15 f. and 44.

283 See a very good description in OXAGILE.

284 Boehm (1988), 64 ff.

285 Sommerville, 48, 112 ff., 256 f. and 572.

tition, particular parts of the software may be reworked and improved after implementation through reviewing.[286] Classic maintainance is automatically done while other new modules are spirally developed.

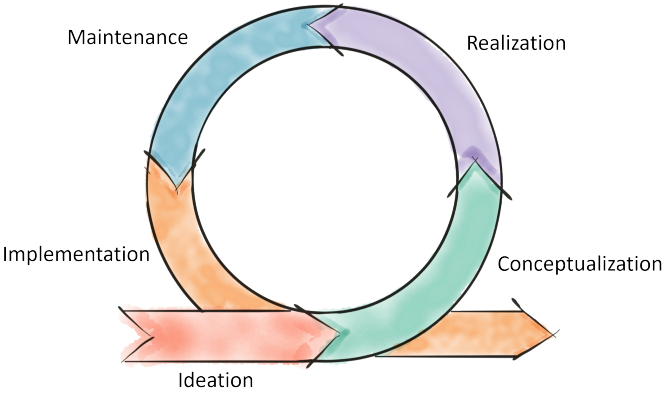


Figure 4. Iterative Development (Source: own illustration).

177 *Incremental* development, on the other hand, refers to the circumstance where different software parts, components, single modules or increments are developed at different stages of the project, which can then be integrated flexibly into the system of modules, once they are completed.[287] The developer, as illustrated in Figure 5, provides successive deliverables or increments, lines them up and connects them to one another. These single modules are potentially releasable. Once one feature is finished – and either directly released or held back for later global release – the developer then works on the next module or feature, until the whole product is completed.[288]

286 Techopedia, "iterative and incremental development", available at <https://www.techopedia.com/definition/25895/iterative-and-incremental-development> (retrieved September 6, 2021); ISO/IEC/IEEE 24765:2017-3.2141, with reference to ISO/IEC/IEEE 26515: 2011, 4.7; Cockburn, 27 and 28; Larman/Basili, 48 f.; Ewusi-Mensah, 76; Becker et al., 75.

287 Techopedia, "iterative and incremental development", available at <https://www.techopedia.com/definition/25895/iterative-and-incremental-development> (retrieved September 6, 2021); Cockburn, 27 and 27 f.; Larman/Basili, 47 ff.; Sommerville, 48 ff.

288 Maynard, 6 and 16 ff.; Koreimann, 165 f.; ISO/IEC/IEEE 24765:2017-3.1917; for more information, see also Larman/Basili.

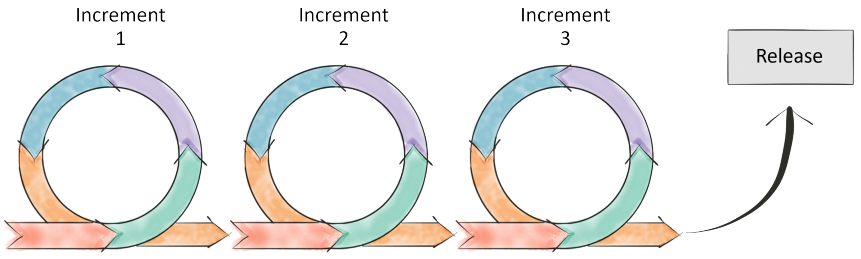


Figure 5. Incremental Development (Source: own illustration).

Agility, as the third property of spiral development, does not represent a descriptive property in a narrow sense, but instead refers to a specific project framework method[289], originally constructed for software development, but today used in project management worldwide. Agility represents a specific subspecies of iterativity and incrementality, as it includes conducting frequent reviews of the development process and its results as well as adapting the process to new or changed needs and practical wishes. But agility also involves a specific form of project organization that builds upon a close-mesh, cross-functional team structure, constant communication and close collaboration between the development teams and the business side.[290] The agile development method that is currently most discussed in the specialist literature is called *Scrum*.[291] In this project management framework, a Scrum team develops each module or increment within a short duration of a few weeks within a so-called *sprint*.[292] Schwaber describes a sprint as a "set of development activities conducted over a predefined period".[293] A sprint usually lasts for one to four weeks, depending on how complex the particular problem is that needs

178

289 The theory and key principles of the approach are described in the Agile Manifesto released in 2001.

290 Techopedia, "agile development", available at <https://www.techopedia.com/definition/13564/agile-software-development> (retrieved September 6, 2021); ISO/IEC/IEEE 24765:2017-3.119, with reference to ISO/IEC TS 24748-1:2016, 2.4, and ISO/IEC/IEEE 26515:2011, 4.1.

291 The methodology is attributed to Ken Schwaber and Jeff Sutherland, who designed the Scrum method following an approach presented by Hirotaka Takeuchi and Nonaka Ikujiro in their paper, calling it the 'Rugby approach' (TAKEUCHI/NONAKA). See the first written description of the Scrum method in SCHWABER and SOMMERVILLE, 75 ff.; see also the original Scrum Guide in SCHWABER/SUTHERLAND.

292 ISO/IEC/IEEE 24765:2017-3.3637, with reference to ISO/IEC/IEEE 26515:2011, 4.9.

293 SCHWABER, 131; see also SCHWABER/SUTHERLAND, 9.

to be solved, the risks that are related to it and how closely the increment is developed to the surrounding technical environment.[294] Within a sprint, a module or its increment is developed, integrated as a deliverable, reviewed and adjusted. Scrum therefore is not only iterative, but also incremental, working gradually from one step to the next.[295] The processing status of each module or "story" is captured on the Scrum Board. Thus each story moves from left to right as development progresses (see arrow in Figure 6).

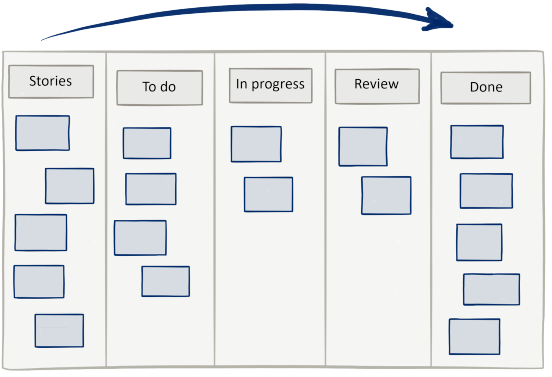


Figure 6. An Example for the Structure of a Scrum Board (Source: own illustration).

179 The spiral development model can also be explained with the help of bridge construction (Figure 7). Again, the idea is to build a bridge across a small creek and connect the two riverbanks. Imagine the constructor does not know whether it is worth crossing the creek – what is on the other side? Does it offer a better way home? The constructor is not looking for a solution to every possible weather situation, nor do they wish to cross the creek with a means of transportation. Instead, the constructor's sole goal for this first attempt is to be able to cross the creek without getting wet. No sophisticated material is available. Only a narrow but long and stable wooden plank is at hand. The constructor places the wooden plank over the creek and thus is able to cross it. If the constructor later decides that they would like to use this path on a regular basis, they will aim to enhance and further secure the crossing path. In a first iteration, the constructor adds another plank to the bridge. This means the crossing is able to handle a bigger load, for example him or her riding on a bi-

294 See SCHWABER, 131; SOMMERVILLE, 85 and 86 f.

295 SCHWABER/SUTHERLAND, 4.

cycle. In further increments, new tasks, such as enabling a greater number of people to cross the bridge or a different type of vehicle, are tackled and solved. With every sequence, using nails, screws, mortar, bricks and so on, the bridge is revised, refined and secured, and thus able to tackle another problem. The basic principle of this approach is that each new stage can build on the previous set of (refined) increments. This means that the constructor meanwhile is able to cross and use the bridge in its respective status (or stage), and profit from each additional module or increment that is embedded, until the bridge has reached its final version.[296]

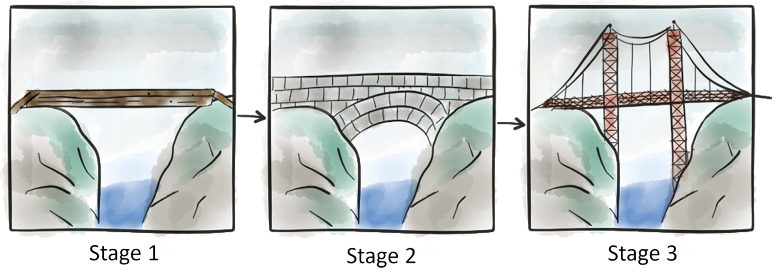


Figure 7. Bridge Construction According to the Spiral Development Approach (Source: own illustration).

There are four main benefits in using the modular or spiral approach compared with a linear one. First, the stages within the development process are transparent and potential difficulties can be detected faster. The process involves regular reviewing, auditing and testing processes, which increases the quality and applicability of the modules before they are implemented.[297] Further, the method is particularly flexible for later changes or adaptations in case the technical or organizational requirements of a project change or have to be adapted.[298] As Scrum works with increments that are developed iteratively, the exact planning for a new module is therefore only made for the next project phase and one goal at a time.[299] As only smaller modules are used, the changes remain on a smaller level and only seldom concern the whole structure. As the final program consists of several individual units, maintenance is simplified. An 180

296 See Caroli.

297 See for both: Schwaber/Sutherland, 4; for both benefits of agile approaches see Belady/Lehman, 232 f.; Singer, 62 ff.

298 See Schwaber, 132.

299 For more information, see Foegen, 112 f.

engineer can simply work on a single block without affecting the remaining ones.[300] One particular benefit of the agile method, is that we obtain a first *prototype* or *first deliverable* as soon as the first module is finished. This can then be shown to the customers or potential investors.[301] The interactive approach allows users and potential customers to be consulted quite early during the development process, instead of presenting or releasing only the final version. This approach further ensures that failing ideas can be detected earlier without the whole computer program having to be programmed.

181 On the other hand, when the spiral approach is followed, it becomes more important to connect the individual modules in the right way. According to Koreimann, the assembled product should not represent a simple chain of implementations but rather a symbiosis and a harmonious ensemble.[302] The engineer should also keep a close overview of the whole project in order to ensure its quality.

3. Continuous Delivery

182 Continuous delivery represents a newer trend in software engineering that has only recently been described in specialized literature and papers.[303] According to this approach, single modules of programs such as add-ons or updates are produced and released within short cycles.[304] Once released, the finished software version is enhanced and altered continuously within its existing mantle while the user is already able to use it. This trend profits from the in-

300 Singer, 63.

301 See also Schwaber, 122; Singer, 33 ff.

302 Koreimann, 165 f.

303 This section of the chapter was added after I had completed the interview series in 2015 and detected that the software development literature I had previously read during document analysis had not covered the continuous delivery approach. Needing to close this knowledge gap in classic computer science literature, at the time this section was compiled, only three papers were found that described the development method from a scientific perspective. Today, the online search engine of the Association for Computing Machinery (ACM) provides around 30,000 search results in its library for the term "continuous delivery", most published in the years 2016-2018 (see online at <https://dl.acm.org/results.cfm?query=continuous delivery&Go.x=0&Go.y=0>, retrieved on July 29, 2019). The ISO Standards have not yet integrated the continuous delivery approach (see ISO/IEC/IEEE 24765:2017).

304 Chen, 50.

creasingly popular approach of releasing computer programs online on servers – partially within the software-as-a-service model – through which actualized versions of computer programs can be distributed directly, and thus faster, to the end users. The main aim is to automate the distribution process of the commercialized computer program by using the direct link to the product.[305]

Working with continuous delivery usually involves four main steps: First, every new code line that is developed is quickly *incorporated* more or less directly with the help of compilers.[306] The element is *tested* and analysed. Then it has to be *integrated* into the previous environment, the software's primary working base. When the program is finished and integrated, it is *released for deployment*, which is when the user can get access to it.[307] 183

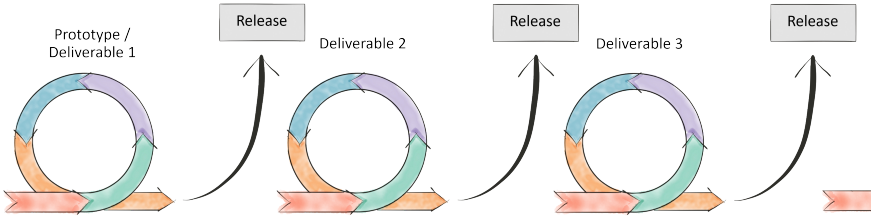


Figure 8. The Continuous Delivery Approach (Source: own illustration).

Building an analogy for the continuous delivery approach with the classic bridge construction is, admittedly, rather difficult. In theory, it would involve that, the first time a heavy truck arrives and needs to cross the bridge, its constructors would increase the bridge's load-bearing capacity with small amendments or improvements with minimal interruption for other users crossing the bridge at the same time. This would necessitate the bridge having some kind of observation system or technical sensors that monitor every section of the bridge and are able to react automatically. Instead of an online system, there would be an attendant who was always present and equipped with every tool he or she might need to react promptly (including the crane in Figure 9). With- 184

305 Linthicum, 8.

306 Humble/Farley, 110 f.

307 See more explicitly in: Linthicum, 6.

out hindering existing users, the attendant could make the appropriate changes, such as strengthening the bridge's beams to withstand the additional weight load of the truck.

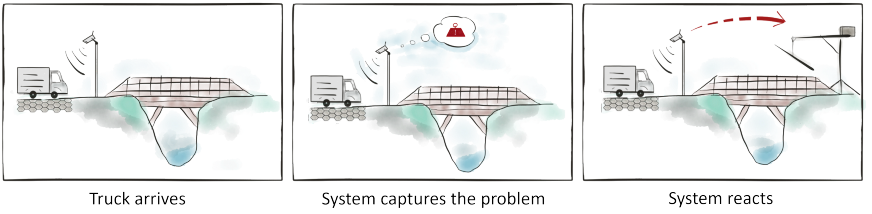


Figure 9. Bridge Construction According to the Continuous Delivery Approach (Source: own illustration).

185 The continuous delivery approach is observable in the habits of many software engineers all over the world. However, Humble and Farley in 2011[308] and Chen in 2015[309] were among the first to describe it in a scientific paper. Chen described in his paper that releasing new software versions every couple of months would artificially delay the process, reducing potential earnings that could be achieved with faster releases. The valuable simple feedback of customers could only be incorporated after a certain amount of time, although small changes would increase the usability of the program.[310] Similarly, Belady and Lehman emphasized in their study that software does not struggle with physical decay like classic inventions do.[311] Instead, computer programs profit from their flexibility. They can be modified or adapted continuously to advance skills, new insights or opportunities. This enables publishers to maintain and enhance their products.[312] As the product can be constantly adapted to new incidents, the quality of the previous version is enriched. The barriers between the software development and the product consumption by the end user can therefore be reduced.[313]

308 Humble/Farley.
309 Chen.
310 Chen, 50 and 53.
311 Belady/Lehman, 228.
312 Belady/Lehman, 227 f.
313 Linthicum, 6; Humble/Farley, 108 f.

According to Chen, there are six main benefits of working with the continuous delivery approach:[314] First, the time to market[315] is accelerated; second, the product can be optimized immediately; third, through automation, the productivity and efficiency can be improved; fourth, automated testing decreases the risks associated with regular deployment processes; fifth, through all these measures, the product is qualitatively enriched as fewer errors occur; and finally, the customer is more satisfied because he or she receives a better product or service. 186

The business value of the continuous delivery approach is therefore rooted in the possibilities to immediately distribute new computer programs.[316] The demands of the customers can be captured and met within less time. This provides a strategic advantage, and ultimately a competitive advantage.[317] The release also becomes less dependent on scheduled intervals. 187

III. Software Project Management and Commercialization

The following section discusses an area that is important for managing a software project economically and commercializing a product or service on the market. It focuses on summarizing and outlining those principles that are relevant for the research study and discussion of the findings. First, a brief overview of the economics of software engineering is provided in order to illustrate some of the economic principles. The next section is about project management and what factors are important in administering software products. Finally, I offer a short introduction to the currently most significant approaches to commercializing software. 188

A. Brief Overview of the Economics of Software Engineering

From an economic perspective, the software engineering industry shows some particular characteristics. The following overview gives a short and simple introduction to the topic for those who are unfamiliar with the economic features of program development, outlining the most important terms and characteristics. 189

314 CHEN, 52 ff.

315 For a definition of the term 'time to market', see later N 198 ff.

316 LINTHICUM, 10.

317 See also HUMBLE.

190 To start with, software is a *dematerialized, non-competing and ubiquitous* good. This means that several people can use the software products at the same time, without it having a reciprocal disruptive or restrictive effect. This is possible because computer programs represent a special form of immaterial intellectual information: whether they are provided on a physical carrier or solely digitally (online), their economic quality is to a large extent irrespective of their embodiment, as their main merit lies in the way the data is processed (their behaviour).[318] As it is not bound to a particular physical form, the software itself can be traded and distributed simpler and faster.

191 The evolving possibilities of technology, and the Internet in particular, over the last twenty years have enabled the facilitated distribution and reproduction of digital goods as computer programs. With the rise of computer networks, *storing and processing data has become less expensive*. As a consequence, originally analogue processes have been increasingly converted into digital ones, representing the initial impulse for what we today call digitalization.[319] While it was economically *rational* to restrict the digital implementation of technology ten years ago, as storage media and data processing were much more expensive, today it is prudent to invest more time and money into digitalization and data mining.[320] As the market shows rapid technological advances, the *pace of product launching* has increased accordingly.[321] As data storage becomes less problematic, faster development processes and providing more and better-performing features of high quality has become more important.

192 Software in general is quite cost-intensive to develop and produce. However, once developed, a set of computer programs can be easily reproduced digitally and at almost no extra cost.[322] This leads to higher scale earnings, as the average cost per copy decreases with every one sold.[323] Software therefore shows *high economies of scale*. The ratio between development costs and revenue from licensing indicates that the software market holds a lot of potential for profit.[324] At the same time, digital information is *distributed much faster* than analogue goods. The positive aspect of this is that a computer program can be

318 For the complete abstract, see RAUBER (1988), 59; BORNHAUSER, N 34 f.; WOESTEHOFF, 100 f.

319 For more information on the technical process of digitalization and its legal implementation, see BORNHAUSER.

320 This issue is also addressed in one of my earlier papers: MURER/ZURMUEHLE.

321 Same interpretation in: U.S. CONGRESS (1992), 9.

322 BOECKER, 75, with further references; LEMLEY/BURK, 90; BORNHAUSER, N 37.

323 See BOEHM (1981), 189 f.; BOECKER, 75, with further references.

324 LANDES/POSNER (1989), 349; KOOTHS ET AL., 20; KATZ/SHAPIRO, 100.

delivered faster to the final user and is thus spread more easily across the world. This opens entirely new market areas. The downside of a faster and facilitated distribution via online servers is that the software companies have to provide a cost-intensive infrastructure to serve the online users. Furthermore, it becomes almost impossible to prevent third parties from utilizing the product. Free-riding is a big problem in software engineering, thus controlling and enforcing commercial rights consequently becomes much more difficult. The associated preventive measures tend to be only marginally useful, and pricey.[325]

As mentioned, there are several characteristics of the software market that should be kept in mind. As software is a non-competing, ubiquitous good, the same development can be offered to several people at the same time without interference. As the possibilities for storing and processing data have made tremendous progress over the last few decades, the economies of scale and possibilities to distribute software have increased. Due to the better availability, software has become better accessible for everyone. This has made the use and production of software more attractive but has also increased the amount and pace of software consumption. 193

B. Software Project Management

According to the International Standardization Organization, a software *project* involves defining an objective, delimiting its scope from other tasks, developing an adequate organizational structure and providing sufficient resources in terms of financing, staff and time.[326] The result of the project is a software product that solves a technical problem through a particular behaviour. 194

This definition suggests that a software project requires a well-planned procedure and reasonable use of the available resources. We can assume that project management has a great influence on the quality of the produced technical good. In order to maintain a positive outcome with the project, it is preferable that one person formulates achievable goals and keeps a clear overview. This *project manager* determines and assigns the organization, means, techniques and responsibilities involved in the administrative 195

325 U.S. Congress (1992), 185; Fisher, 19.

326 ISO/IEC/IEEE 24765:2017-3.3152, with reference to ISO/IEC/IEEE 15939:2017, 3.33; ISO/IEC TS 24748-1:2016, 2.35; ISO/IEC/IEEE 15288:2015, 4.1.33; see also DIN 69901-5:2009.

process.[327] The project manager has to make important decisions about the cornerstones of a software product, how it is developed and which strategy is used to commercialize it. These decisions are partially guided by limited, assigned resources.[328] His or her job consequently involves planning, scheduling, budgeting and creating rational and comprehensible structures, while considering the available resources, the timeline and the complexity of a project.[329] It is the manager's responsibility to ensure that the conceptual framework and purpose of the product meet the business idea of the company and that the individual elements, steps and components of a network match, technically, organizationally and strategically.[330] He or she is also responsible for quality assurance. In larger software companies, these tasks may be subdivided and assigned to several different people or entities, such as a product manager, software architect or marketing manager.

196 During the last two decades, information and communication technology have become better applicable and more affordable not only for businesses but also for private households. One of the main goals of a project manager is to satisfy the needs and expectations of the users. Keeping up with the high-paced and dynamic demands consequently represents a great challenge for every software project. Millson and Wilemon emphasize in this context that the user today expects the products to exhibit flawless quality, provide added value at lower prices and to become available within a shorter time.[331] "Only the best is good enough".[332] In Offut's study, he suggests that the main three characteristics that are evaluated by users to measure the quality of software are its *reliability*, its *usability* and *how secure a computer program is* in terms of data protection etc.[333] All three of these elements have thus become significant for the development of computer programs and managing its processes.

---

327 ISO/IEC/IEEE 24765:2017-3.3178, with reference to ISO/IEC 26514:2008, 4.39; see also DIN 69901-5:2009; Bauknecht/Zehnder, 282 f.; Becker et al., 241 f.

328 Boehm (1981), 23 f. and 727 ff.; Becker et al., 241.

329 Smith, 180; Boehm (1981), 30; for detailed reports on which factors influence a software development process and its success, see: Standish Group and SwissQ.

330 Krueger/Pfeiffer, 21 ff.; Koreimann, 159 f.

331 Millson/Wilemon, Foreword, x.

332 This is the slogan of the Lego Group. Lego has recently become one of the most important technology companies worldwide, being named number 82 in the world's top 100 brands for 2015 (see <https://www.lego.com/en-us/aboutus/lego-group/the_lego_history/1930> [retrieved September 6, 2021], for ranking, see CNet).

333 Offut, 27.

Consumer needs and money issues are regularly considered during the development process. The software product should, for example, show high *scalability* in order to handle larger work volumes simultaneously.[334] Network integration and maintenance have become an intensive cost factor that is manipulated positively during conceptualization and coding. Managing the software, therefore, starts as soon as the development process is organized and structured, as unnecessary reworks that cost a lot of money can be anticipated and partially avoided. 197

The *time-to-market* is regarded as a major factor in software management and is often used as a buzzword. It expresses the duration of time that is required to develop a product from the idea to the final product.[335] Time-to-market refers to the fact that the software market has become faster in pace and that competition has increased. Whoever brings additional value to customers first, gains their share-of-wallet. To maintain a company's competitiveness, the software developer tries to adapt the launching pace of the software to the demands of the market and releases the software, or only selected modules of it, sooner and faster. The development process is thus shortened or accelerated.[336] Offut believes that time-to-market is currently one of the key business drivers of the industry.[337] The downsides of a faster development velocity are higher product costs, greater expenses for development and either less performance of the software product, as the product may not be refined enough, or a smaller set of features.[338] At the same time, Cohen, Eliasberg and Ho suggest that optimal software engineering should "concentrate efforts on the most productive stage".[339] The developer should try to achieve an adequate balance between all the goals in order to mitigate the effects of potential objective conflicts.[340] 198

The above-mentioned factors suggest that software project management is a very complex issue. A project manager has to audit several very different sub- 199

334 Offut, 28.

335 The Business Dictionary, "time to market", available at <http://www.businessdictionary.com/definition/time-to-market.html> (retrieved July 27, 2019).

336 This can, for example, be achieved through faster development, minimizing schedule variation, improving agility, avoiding mistakes and reworks, improving productivity, overcoming sagging revenues or market share and through sticking to schedules (for more information, see Smith, 176). See also Bock, 223 f.

337 Offut, 29.

338 Smith, 174 f.

339 Cohen/Eliasberg/Ho, 175 and 184.

340 Cohen/Eliasberg/Ho, 174 and 184.

ject areas at the same time, including organizational, financial and time questions.[341] The expectations of the user and buyer have increased dramatically. A dynamic industry and newly emerging trends such as time-to-market and shorter launching paces keep presenting new challenges. The profile of a software project manager consequently has become very versatile. These features of project management should be further evaluated and considered from the legal aspect.

C. Commercialization

200 Commercialization is about distributing the software to the customers. Depending on the market being served and the financial and organizational structure of the publishing company, different strategies for economic management and legal commercialization are followed.

1. Classic Software Commercialization

201 Intellectual property law grants particular exclusive rights to developers, including some commercial rights for economic purposes to earn back the investments made. In practice, the issuer's exclusive rights are often used to make the software product available to third parties. The exact terms of the allowable utilization are regularly defined in contracts. The granted rights widely vary in their scope, function and exclusivity.

202 There are three commercialization models that seem to be used frequently for distributing software: assignments, licensing and service agreements:

- If a right is *sold* or otherwise *assigned*, the right holder surrenders his/her rights of use and passes them on to somebody else.[342] This often occurs with commissioned work and appointed product development by third-party contractors and consultants.[343]
- Through *licensing*, the legal ownership over a good is not assigned but instead the licensee obtains the temporary permission of the right holder (licensor) to make use of the good in question in a particular, contractu-

341 See detailed reports on which factors influence the software development process and its success, in: Standish Group and SwissQ.

342 Hilty (2010), N 288; Woestehoff, 90 ff., particularly 100 f.

343 Nimmer/Nimmer (2014), N 27-36 and N 27-45.

ally agreed way.[344] There are two main forms of licences available: exclusive[345] and non-exclusive rights.[346,347] Difficult to distinguish from licence contracts are leases, in which the user usually has proprietary rights similar to ownership.[348] The rights of use often go further than in traditional licensing, which is probably also the reason why leasing plays a lesser role in practice and is not dealt with in more detail below.

– A newer form of software commercialization is so-called *software as-a-service*. Based on a client server system, the customer is usually granted online access to a particular software product on a server.[349] In return, he or she pays service operation and maintenance fees.[350]

Licensing and software as-a-service are usually mixed in practice. 203

2. Open Source and Free Software in Particular

The Open Source Movement developed in the 1990s and became a large and important trend in software commercialization with the rise of the Internet.[351] Open Source refers to the idea that know-how of software engineering will be exchanged, shared and passed on within a community. We still lack a generally accepted definition of the term Open Source. However, according to the Open Source Initiative, Open Source refers to non-discriminatory and unhindered 204

344 Hilty (2010), N 288; ISO/IEC/IEEE 24765:2017-3.2213 and ISO/IEC/IEEE 24765:2017-3.3820, with reference to ISO/IEC 19770-5:2015, 3.41; Hilty (1997), 139 f.; Woestehoff, 110 ff., particularly 117 f. and 120 ff.; Nimmer/Nimmer (2014), N 27-3 f.; Lemley et al., 227 ff.; Commentary to the German UrhG (Loewenheim/Spindler), § 69a ff. N 59 ff.

345 Exclusive rights refer to the situation where only one party is allowed to make use of the creation, instead of several.

346 Non-exclusive rights imply that several independent parties obtain the right to use the creation in question in a particular way.

347 Mentioned in Art. 62 para. 3 Swiss CopA, § 31 para. 1 sentence 2 and para. 3 German UrhG, 17 U.S. Code § 101; see also Froehlich-Bleuler for indirect use in licensing.

348 Woestehoff, 163 f.

349 For more information on the term server, see N 128.

350 Buxmann/Hess/Lehmann, 500; Kooths et al., 32 ff.; Sommerville, 499; see also description in ISO/IEC/IEEE 24765:2017-3.3708.

351 Harison, 78 f.; Cowan/Jonard, 515 and 529 f.

access to the source code, free and unrestricted distribution of the program, and an obligatory authorization to modify the program and distribute the derived works.[352]

205 The goal of Open Source is that everybody can take advantage of comprehensive use of the software, without being hindered by the proprietary demands of a copyright holder.[353] It represents a political statement. Behavioural economists qualify the Open Source community as altruistic, sharing their know-how and investments with other partially unknown users. On the other hand, there seems to be high-level expertise provided in the available solutions.[354]

206 Open Source does not represent a legal model of its own but instead builds on the traditional framework of proprietary intellectual property rights and licence agreements.[355] The existence of copyrights is a prerequisite for the existence of Open Source. However, in contrast to proprietary software commercialization, in Open Source the copyright is not used to earn back the investments but to steer a free know-how transfer instead:[356] What right holders make available free of charge under an Open Source licence cannot be monopolized by others.[357] Instead, every user has to guarantee contractually that he or she will share the results they achieve on the basis of the computer program for free with the community and that other users will be allowed to work on them and provide changes and improvements under so-called public licences or copyleft.[358] There are a wide range of Open Source licences available on the market. They are partly standardized by the Open Source Initiative, which has developed certain minimum requirements for Open Source soft-

352 For further information on the Open Source Initiative, see their webpage at <http://opensource.org/osd-annotated> (retrieved September 6, 2021).

353 Koglin, 1; Spindler, 1; Zirn, 156 ff.; Schwabach, 186.

354 Harison, 78 and 81 f.

355 Spindler, 2 f. and 25 f.; Koglin, 25; Boecker, 190; Straub (2011), N 638 ff.; Hilty (2014), 293 ff.; Schwabach, 186; Commentary to the German UrhG (Loewenheim/Spindler), § 69a ff. N 28 f.

356 Schiffner, 104; see Boecker, 190, for further information about the relationship between copyright and Open Source.

357 Straub (2011), N 689.

358 Metzger/Jaeger, 431; Harison, 81; Lessig, 764 f. The legal qualification of Open Source distribution is not beyond controversy. It is unclear to what degree an author can resign his or her statutory rights of exclusive consent (for more information, see Koglin, 2 ff. and 42 ff.).

ware within the framework of the Open Source definition.[359] The strictest form of Open Source licence appears to be the *Gnu General Public License*, which was created by the supporters of the *Free Software Movement* around Richard Stallman.[360] Like classic Open Source, the term free software usually refers to the permission to use, copy or spread software, either in its original form or in an adapted version. It also requires that the source code be disclosed.[361] However, the user of the modules is usually subject to stronger duties.

The most famous example of Open Source software was probably the operation system Linux that was established by Linus B. Torvalds and today is estimated to run on over 40 million computers worldwide.[362] For proprietary software, Open Source models are often described as unattractive because every user is allowed to analyse and reproduce the technical solutions in the source code, and to share them. As the comprised know-how is disclosed for free, the software can hardly be sold against payment.[363] Nevertheless, proprietary software developers have started to embed parts of the Open Source model in their corporate commercialization strategy, to profit from the know-how and community thoughts.[364] In addition, *dual-licensing* solutions are increasingly being offered on the market. Within this framework, the same software is distributed under both a proprietary licence and, simultaneously, an Open Source licence.[365] The distribution models in the Open Source are still evolving and, as shown later, Open Source still has a significant effect on the software engineering market.[366] 207

[359] For more information, see official Open Source Initiative webpage at <https://opensource.org/osd> (retrieved September 6, 2021).

[360] For more information on the principles of GNU, see the official GNU webpage at <https://www.gnu.org/copyleft/> (retrieved September 6, 2021).

[361] BECKER ET AL., 88 and 91; SPINDLER, 2 f. and 9 ff.

[362] There are few reliable statistics available for potential user numbers or installed applications. The most comprehensive report was published by WIKIMEDIA. The numbers vary, but it is evident that the Open Source community has gained ground over the last two decades.

[363] KOGLIN, 8.

[364] See for example Google's cooperation with the Open Source community: <https://developers.google.com/open-source/> (retrieved September 6, 2021).

[365] SPINDLER, 16 f.

[366] See later N 421 ff.

Chapter 4: Status Quo of Legal Software Protection

208 This chapter summarizes the legal foundation for my research and builds on a review of the available literature, conducted over several months. It provides an idea of how legal protection for computer programs is designed today, for which elements in a computer program it might apply, and where potential problem areas might lie. The following illustration is not intended as a complete presentation of software protection law in these fields. Instead, the chapter focuses on basic topics I consider crucial for an understanding of the interview findings and constructional thoughts in the subsequent discussion. The legal illustration therefore focuses on the essentials regarding the legal scope of software protection in the observed jurisdictions.

209 The chapter starts with a brief overview of the available legal measures for protecting software. As some of the legal institutions are not within the scope of my thesis, they only serve to provide comprehensiveness. It will be explained why intellectual property law is important for software and what functions it serves. In a next step, the international context of intellectual property law is outlined in order to point out what should be considered beyond national legislation. Finally, a systematized illustration introduces the basics regarding the scope of protection for software in patent law and copyright in three different observed jurisdictions: Switzerland, the European Union and the United States. As intellectual property law has been highly standardized on an international scale, the principles in copyright and patent law in the three jurisdictions are very similar, leaving only little room for individual provisions in national statutes. They are examined together, outlining their common structures, the most important prevailing legal approaches as well as some diverging points of interest for the subsequent legal discussion.

I. The Legal Institutions for Software Protection

A. Introduction to Legal Software Protection

210 Within the European Union, Switzerland and the United States, the protection of computer programs is designed as a *hybrid* concept. There is no one legal

institution that covers every aspect of software in one model. Instead, there are various legal institutions that each cover a different aspect and are partially overlapping:[367]

- First, computer programs can be protected under copyright and patent law. These two institutions are the most relevant in the literature and jurisprudence and thus are the two that are analysed most comprehensively in the present study;
- second, industrial design protection and trademark law are partially accepted as protecting particular program components;
- third, some legislations provide for so-called utility models – a lighter version of classic patents to protect subordinate inventions;
- fourth, there are the classic institutions of competition law such as unfair competition, including in particular protection of trade secrets and measures against others taking undue advantage of your achievement;
- fifth, the hybrid of the property assigning IP and punitive unfair competition law is complemented with the classic rules of contract law.

All of these institutions protect particular aspects or whole components of software development and completed software products. Consequently, together they represent the commonly followed protection model for software within the EU, Switzerland and the United States. 211

B. Brief Overview of the Available Legal Protection Measures

What follows is a brief overview of the different legal measures that are available and acknowledged for software protection. It does not claim to be exhaustive, nor does it claim to provide the only possible interpretation of the illustrated complex problems. In this context the legal software protection is a hybrid of very different institutions and each field provides selective protection for particular software components. It will be shown that the different legislations all greatly rely on copyright and patent law to protect components of software products. These topics are therefore elaborated on in two separate sections (*sections IV and V* of this *Chapter 4*). The explanations in this overview will therefore focus on institutions for legal software protection other than copyright and patent law. 212

[367] Due to their lack of relevance for the protection of digital software products, microelectric semiconductor products (hardware chips) and their protection in topography law will not be addressed in this thesis.

1. Intellectual Property Law

a) Patent Law

213 A patent represents an intellectual property right that protects new, non-obvious and industrially applicable innovations. It comprises an exclusive right of the patentee to forbid others to use his/her invention, no matter in what way the invention was implemented. The right is granted by a particular registration authority for a determined territory in exchange for public disclosure of the invention. Computer-related inventions are only partially open to patent protection, e.g. implemented and applicable algorithms, to some extent particularized business methods, and according to present opinion, also functions and features (disputed). For more information see *section* IV of this chapter.

b) Copyright

214 Copyright is intended to protect original intellectual creations of art and literature. Its protective scope evolves automatically as soon as the creation has fulfilled the basic protection criterion. It offers the copyright holder a set of exclusive commercial and moral rights. Copyright represents the main form of protection for computer programs in the EU, Switzerland and the United States.[368] Components that are eligible for copyright protection are particularly literary works, such as the source code (including its structure and organization), the machine code as well as some visual elements such as the graphic user interface and, according to the here presented opinion, also visual elements in the look-and-feel. It further entails one-to-one protection for whole software products and services and for the provided documentation. For more information see *section* V of this chapter.

c) Industrial Design Rights

215 Industrial designs represent aesthetic or ornamental combinations of lines, contours, colours and surfaces that are contained in a useful product.[369] These

368 See for example: Straub (2002), N 4; Govoni, 570; Lehmann (1988), 2420; Lehmann (1988), 2420; Hilty (2010), 114 f.; Hilty/Geiger (2015), 615 ff.; Jersch, 192; Waltl, 1 f; see also discussion in Scotchmer (2006), 83 f.; Contu, 59; Commentary to the German UrhG (Loewenheim/Spindler), § 69a ff. N 8; due to its technical function, copyright for computer programs has often been criticized, see for example: Hilty (2010), 114 f.; Hilty/Geiger (2015), 615 ff.; Boecker, 27; Scotchmer (2006), 83 f.

369 Art. 1 Swiss DesG, § 1 para. 1 German DesG, 35 U.S.Code 171.

combinations are protectable under design rights, if they are new and exhibit particularity.[370] Design rights have to be granted formally and require registration with the responsible authority. The owner of a protected industrial design may prevent third parties, who do not have his or her consent, from commercially making, selling or importing articles bearing or embodying a design that represents a copy of the protected design.[371]

It is still very controversial as to whether software components are protectable through design rights. In many statutes computer programs are explicitly excluded from the subject matter.[372] This is rooted in the fact that classic design rights are intended for visual and tangible objects.[373] In many jurisdictions, visual elements that merely result from a technical function of a product are not eligible for protection, as it is the technical aspects rather than the aesthetic characteristics that are in focus.[374] The question of what may be considered as 'serving a technical function' in a software product is very difficult to answer. Some legislations do not specifically regulate for this problem but instead leave this decision to the responsible authority and courts. Consequently, different jurisdictions answer this problem in different ways. 216

Software is based on abstract numeric and instructive commands that have to be expressed in a written form, such as the source code or the machine code. This aspect of a computer program serves a functional purpose rather than an aesthetic or literary one, from the perspective of a design right. It also lacks a visual design concept.[375] It is therefore excluded from design right protection. Design right protection is more relevant for the visual components of a software product, such as the graphic user interface or the look-and-feel. Although these two elements are not tangible they can be graphically represented and have perceptible lines, shapes and colours, based on a design concept. While they can be expressed in a visual form, they are also used to 217

370 Art. 2 para. 1 Swiss DesG, § 2 para. 1 German DesG, 35 U.S.Code 171.

371 Art. 26 TRIPS Agreement.

372 See particularly the understanding of the European Union in Art. 1 lit. b of the Directive 98/71/EC of the European Parliament and of the Council of 13 October 1998 on the Legal Protection of Designs, that probably also influenced the Swiss understanding of Art. 1 Swiss DesG (Orell Fuessli Commentary to the Swiss DesG, Art. 1 N 81).

373 Orell Fuessli Commentary to the Swiss DesG, Art. 1 N 10; DUTFIELD/SUTHERSANEN, 171 ff.

374 See for example Art. 4 lit. c Swiss DesG, § 3 para. 1 sect. 1 German DesG; 35 U.S. Code § 171 speaks of 'ornamental' designs. In *Richardson v. Stanley Works, Inc.*, 597 F.3d 1288 (Fed. Cir. 2010) it was established that a non-functional characteristic must be shown.

375 See Orell Fuessli Commentary to the Swiss DesG, Art. 1 N 4; Staempfli Commentary to the Swiss DesG, Art. 1 N 32.

communicate with the users. Therefore, although a graphic designer tries to create this medium of communication in a visually pleasing way, the visual expression always also has a functional purpose because the software product is an economic good. The visual and functional aspects of computer programs consequently lie close together. Whether the visual elements of software are implied under design right protection or not is widely disputed. Some claim that every possible visual expression is excluded from design right protection.[376] Others, including the WIPO, several Swiss representatives and U.S. courts and authorities have decided that design right is applicable.[377] Kur, on the other hand, differentiates between the user interface, implemented icons and the look-and-feel of a computer program, stating that the first two would be eligible while the third component should be excluded due to its merely technical nature.[378] As Stigler suggests, what is eligible for design protection in a software product can probably not be determined in absolute terms, but instead should be evaluated for each program separately in order to determine whether the "design is entirely dictated by its function".[379] Likewise, where the functionality predetermines every aspect of the visual interpretation, legal protection under design right should not be allowed. On the other hand, where there is merit in the visual perception, design right protection should be possible. In practice, graphical user interfaces have already been registered as industrial designs.[380]

376 See for example Harison, 185 f.; Art. 1 lit. b of the Directive 98/71/EC of the European Parliament and of the Council of 13 October 1998 on the Legal Protection of Designs, which explicitly excludes design protection for visual expressions in computer programs.

377 The official information on the WIPO home page suggests that "industrial designs are applied to a wide variety of products of industry and handicraft items (...) graphic symbols, graphical user interfaces (GUI), and logos". See additional information on the website of the WIPO on industrial designs; European Guidelines for Examination of Registered Community Designs, 17; Orell Fuessli Commentary to the Swiss DesG, Art. 1 N 82; Straub (2011), N 506 ff.; Straub (2013), N 38.1 (noting that some design rights for user interfaces have already been registered in Switzerland); Marly, N 641; Staempfli Commentary to the Swiss DesG, Art. 1 N 32 f.; see for the U.S.: *Apple, Inc. v. Samsung Electronics Co., Ltd.*, No. 14-1335 (Fed. Cir. 2017), which says that if a user interface is also non-functional and fulfils the requirements, it may be considered as design patentable; for more information and references on the U.S. perspective see: U.S. Congress (1992), 12; Stigler, 238 ff.; Lemley et al., 195 ff.

378 Kur (2002), 663; see also Kur (2003).

379 Stigler, 240; see also Kur (2002), N 32.

380 See for example, for the United States: AcclaimIP; for Switzerland: <www.swissreg.ch>; for Germany: <https://register.dpma.de>.

d) Trademark

A trademark is a sign or a combination of several signs that are capable of distinguishing the goods or services of one company from those of other companies.[381] All signs that can be recognized and noticed can serve as a trademark.[382,383] These signs may consist of words, letters, numerals, figurative elements and combinations of colours as well as a mixture of any of these. Eligible for registration are signs that have a distinctive character.[384] There are no further requirements for protection, only criteria that exclude certain objects from the subject matter. Generally excluded are signs that belong in the public domain and characteristics that are predetermined in the technical conceptualization of the product, including particular essential forms, goods and packaging as well as signs that may be deemed deceptive.[385,386] In order to protect earlier trademarks, there are certain relative grounds for exclusion, for example if the trademark clashes with an earlier registered one because they are identical or similar and are intended for the same or similar goods or services. 218

Trademarks can be registered for different jurisdictional territories, for example for individual countries, within a regional trademark union or at the WIPO on an international level. The owner of a trademark obtains the right to prevent third parties without consent from using identical or similar signs for goods or services that are the same or similar to the object in question.[387] 219

381 Art. 15 para. 1 TRIPS Agreement; Art. 1 para. 1 Swiss MSchG; § 3 para. 1 German MarkenG; 15 U.S. Code §1052.

382 Staempfli Commentary to the Swiss MSchG (Noth/Thouvenin), Art. 1 N 8; THOUVENIN/BERGER, 6/6.1., 1.

383 In the United States, the Lanham Act further protects so-called trade dresses, comprising the total design of a product packaging, including its shape and colour. According to certain authors, the trade dress provisions may also entail the look-and-feel of computer programs. See discussion in LEMLEY ET AL., with further references.

384 Art. 15 para. 1 TRIPS Agreement; Art. 1 para. 1 Swiss MSchG; § 3 para. 1 and § 8 para. 2 sect. 2 German MarkenG;

385 See for example Art. 2 lit. a-c Swiss MSchG; § 3 para. 2 German MarkenG.

386 See discussion regarding Microsoft's 'Windows' trademark for computer programs, in LEMLEY ET AL., 210 f. and THOUVENIN/BERGER, 6/6.2., 1 f., both with further references.

387 See Art. 16 para. 1 TRIPS Agreement.

220 Trademarks are used as a means of communication between a company and their customers.[388] Their distinctiveness is broadly associated with the quality that the public assigns to a good or service.[389] They serve to differentiate and reference the manufacturers or providers of the marked good or service.[390] Trademarks are therefore popular for marketing and in establishing a particular public image of a good or service. In the field of software, the designations of computer programs and ICT services are often protected with trademarks, for instance, the operating system software Linux®, and the internet browser Mozilla Firefox®. The German group SAP, a famous developer of enterprise software, provides a whole list of their goods and services that are currently under trademark protection on their homepage, including their programming language ABAP®, their company name SAP® and the computer business network SAP Ganges®.[391] The Swiss telecommunication provider Swisscom has likewise registered several ICT goods and services, such as Swisscom IT Services®, Swisscom Solutions® and Swisscom Broadcast®.[392] A similar policy may be observed for the U.S. group Oracle.[393] The possibility to protect the designation of a software product or service with trademarks is attractive for managing a company's corporate identity and influencing the public perception of a good or service. Consumers show trust in established and popular brands, which is why, particularly in the computer field, they place significant value on obtaining application programs, hardware components and further equipment compatible with trademark-registered products and known manufacturers.[394] The attentive observer may have noticed that all the alluded to registered trademarks refer to the software product as a whole. The practical use of software trademarks is currently broadly limited to product names of standard software.[395] But in theory other perceptible, e.g. visual, elements are

388 Staempfli Commentary to the Swiss MSchG (Noth/Thouvenin), Art. 1 N 45; THOUVENIN/BERGER, 6/6.1., 1.

389 DUTFIELD/SUTHERSANEN, 139; Staempfli Commentary to the Swiss MSchG (Noth/Thouvenin), Art. 1 N 47.

390 Staempfli Commentary to the Swiss MSchG (Noth/Thouvenin), Art. 1 N 34 ff. and 39 ff.

391 See interesting list of currently active trademarks of the company SAP at <http://www.sap.com/corporate-en/about/legal/copyright/trademark.html> (retrieved September 6, 2021).

392 For more information, see the Swiss Trademark Registry: <www.swissreg.ch>.

393 For more information, see the European Union Trademark Registry <https://oami.europa.eu/> and the U.S. Trademark Registry <www.uspto.gov>.

394 LEMLEY ET AL., 201.

395 See also STRAUB (2011), N 33 and 580; MARLY, 593 ff., with further references; Commentary to the German UrhG (Loewenheim/Spindler), § 69a ff. N 11.

also eligible for trademark protection if they are able to exhibit enough distinctiveness and characteristics, and can stand out from others or functionally pre-conditioned signs.[396] It is conceivable that in the near future graphical user interfaces and visual elements of the look-and-feel in software will also become eligible for trademark protection.[397]

e) Utility Models (Gebrauchsmuster)

Some countries offer so-called utility models or petty patents to protect inventive devices. They are granted in many countries all over the world, including the Czech Republic, Denmark, Finland, France, Austria, Germany, Italy and Portugal.[398] In Switzerland and the United States the utility model is not available. A utility model is a statutory monopoly, similar to a patent, but with a smaller scope of protection.[399] It is therefore also sometimes referred to as the small patent. Just like patents, utility models require novelty, an inventive step and the exhibition of an applicable invention.[400] But different to patent law, they exclude inventive procedures entirely, instead focusing on products and devices.[401] The referential room for the testing of the essential requirements is usually restricted to publicly available written descriptions and established uses of the invention within the jurisdiction in question.[402] Further, the formal registration process does in general not involve any substantive testing of the protection requirements.[403] The utility model therefore represents an untested intellectual property right. According to the WIPO, the main purpose of utility models is to encourage inventors of 'minor inventions' to share their inventions with the public, including improvements of and adaptions to exist- 221

396 See discussion in LEMLEY ET AL., 203.

397 This topic came up several times as a side note in the interviews.

398 See list of current national providers, available at <https://wipolex.wipo.int/en/legislation/results?subjectMatters=2> (retrieved September 6, 2021).

399 MOTT, 233 ff.; DUTFIELD/SUTHERSANEN, 180.

400 See for example § 1 para. 1 German GebrMG; § 1 para. 1 Austrian GMG, Art. L611-11 and Art. L611-14 French IP Code.

401 See for example § 2 para. 3 German GebrMG, § 1 para. 3 sect. 3 Austrian GMG.

402 See for example § 3 para. 1 German GebrMG, §3 Austrian GMG.

403 See § 8 para. 1 German GebrMG, § 18 Austrian GMG.

ing products.[404] For this purpose, the owner is granted an exclusive right for a limited period of time, which allows him or her to prevent others from using the protected device commercially.

222 The application field of utility models is widely restricted by law. Various statutes prohibit the protection of procedures and reproductions that in essence work with (non-tangible) information.[405] For this reason, the use of utility patents is highly restricted for computer programs.[406] However, as an exception, in the Austrian Gebrauchsmustergesetz, the program logic underlying programs for data processing equipment may be protected as a utility model.[407] Apart from this exception, the application scope of utility models is usually limited to cases where the software is combined with physical devices, known as hardware-related software.

2. Unfair Competition Law

223 Unfair competition law involves any action in the competitive market that is contrary to honest practices in the industrial and commercial sphere.[408] It also includes misleading or deceptive behaviour that may affect the relationship between competitors or between a supplier and their customers.[409] Particularly important for the field of software development and commercialization is the protection of trade secrets and the prohibition on taking undue advantage of somebody else's achievements.[410] Further, within a limited scope, the corporate identity of a company implemented in the look-and-feel can be protected under the general clause of unfair competition law, if there is a risk of confusion.[411] Beside these examples, the impact of the general clause for software is likely to be limited. The next section focuses on the first two topics.

---

404 See the WIPO's official online description of "utility models", available at <https://www.wipo.int/patents/en/topics/utility_models.html> (retrieved September 6, 2021).

405 See § 1 para. 2 sect. 4 and § 2 para. 3 German GebrMG, § 1 para. 3 Austrian GMG, Art. L611-2 para. 2 French IP Code.

406 KRIBBER, 18 f.

407 See § 1 para. 2 Austrian GMG.

408 Art. 10bis para. 3 Paris Convention.

409 See general clause in Art. 2 Swiss UWG; § 4 German UWG; 15 U.S. Code § 45 lit. a para. 1.

410 For a broader view of the topic, see WIEBE; MARLY, 237-252; STRAUB (2011), N 522 ff.

411 Due to its limited relevance particularly for software and the restricted scope of this thesis, I will not further elaborate on this point.

a) Trade Secrets

Trade secrets refer to technical and commercial information that guides manufacturing and commercializing activities in a particular way.[412] Any unauthorized use of information which qualifies as a trade secret and is obtained by misappropriation is internationally considered as a violation of the trade secret and therefore as an unfair practice if conducted by a third party other than the secret-keeper.[413] Discovering information independently that qualifies as a trade secret does not fall under the protected scope. It is only if the information is obtained by misappropriation, for example by discovering the trade secret through unlawful means such as hacking, data theft or some form of espionage, or where a third party discloses a trade secret by breaching a contract, that trade secret violation is considered to have taken place. In contrast to patents, the protection of trade secrets does not require formal registration. The legal protection of trade secrets begins the moment that they qualify as sensitive information. 224

In order to be qualified as a protectable secret, information has to fulfil three requirements: First, it has to be secret. It cannot be information considered as general knowledge or readily accessible to people within circles that commonly deal with the kind of information in question.[414] Second, the information must exhibit economic or commercial value, e.g. it is used to work a market.[415] It must be characterized as worthy of protection because there is an interest in secrecy protection, particularly information that is capable of providing a competitive advantage for a company.[416] Third, the rightful secret keeper must have taken reasonable measures to keep the information secret.[417] According 225

412 See definition of the WIPO for "trade secret", available at <https://www.wipo.int/tradesecrets/en/> (retrieved September 6, 2021).

413 See Art. 39 TRIPS Agreement in conjunction with Art. 10bis Paris Convention that guarantees trade secrecy for all member states. See for a national implementation example § 1 German GeschGehG, Art. 4 lit. c and Art. 6 Swiss UWG and 18 U.S. Code § 1839 as well as the UTSA for the United States. If the abuse was conducted by a person who was entrusted with a trade secret see Art. 5 lit. a Swiss UWG.

414 Art. 39 para. 2 lit. a TRIPS Agreement; 18 U.S. Code § 1839 para. 3.

415 Art. 39 para. 2 lit. b of the TRIPS Agreement; 18 U.S. Code § 1839 para. 3; § 1 para 4 sect. i UTSA

416 See definition of the WIPO for "trade secret", available at <https://www.wipo.int/tradesecrets/en/> (retrieved September 6, 2021).

417 Art. 39 para. 2 lit. c of the TRIPS Agreement; 18 U.S. Code § 1839 para. 3; § 1 para 4 sect. ii UTSA.

to Swiss case law, trade secrets are protected as long as the information in question is regarded as 'objectively secret' and is also treated confidentially by its keeper.[418]

226 Trade secret protection plays an important role in software development and commercialization, as both of these processes involve a lot of know-how and high investment. The highly sensitive information that needs to be protected may include certain tactics, programming techniques, system-relevant information or a solution to a certain problem, including information about the logic, structure and ideas underlying a computer program.[419] The elements that are frequently described as particularly sensitive and valuable are algorithms, sensitive parts of the source code and ingenious information about the software concept.[420] As trade secret protection applies until previously secret information becomes public, it regularly precedes registry intellectual property rights, that require adequate disclosure during application.[421] It is worth noting that as information under trade secret protection is not publicly available, it is also not respected in examinations of 'prior art' for patent law.[422]

b) Taking Undue Advantage of Somebody Else's Achievement

227 Apart from trade secrecy, unfair competition law in most countries also forbids a competitor from taking undue advantage of somebody else's achievements. This particularly includes offering imitations of somebody else's work products in products or services, if either (1) no sufficient personal effort was made to copy the original, or (2) if the knowledge to imitate the original was

418 See example for Switzerland: BGE 64 II 170 and BGE 80 IV 22. Taking reasonable measures as such does not represent a separate criterion under Swiss law, but instead is considered when evaluating the keeper's subjective will and the possibilities to keep a secret confidential.

419 WITTMER, 72 f.; U.S. CONGRESS (1990), 9; LEHMANN (1988), 2422.

420 See BOECKER, 112, with further references, particularly regarding the decision of the OLG Celle of April 11, 1989, published in CR, 1989, 1002 ff.; see also discussion in LEMLEY ET AL., 16 f.; see also later discussion in N 526 ff.

421 See also discussion in LEMLEY ET AL., 13 f.

422 U.S. CONGRESS (1990), 9.

obtained in an unrighteous way.[423] Similar to copyright and patent law, the protection is not aimed at abstract ideas, but rather processed concepts in a marketable, economically usable and independently exploitable and materialized form of the working result.[424] Moreover, the provision particularly targets technical reproduction processes, as the original can be copied without creating added value or investing further effort, and the product can be manufactured and offered on the market at a lower cost.[425] One important exception to the general prohibition on taking undue advantage is provided in the United States where 'copycat acts' are permitted if no intellectual property rights are violated.[426]

Also computer programs are regarded as achievements under unfair competition law.[427] Tied to this conclusion, digital copying and offering to stream and download is regarded as a predatory act.[428] Although the possibilities under unfair competition statutes are numerous, its relevance for software protection in practice is usually restricted to the period between patent application and patent granting[429], cases where standard software is copied one-to-one, or if technical reproduction measures are applied to extract know-how from 228

423 See § 3 and § 4 para. 3 in German UWG, Art. 5 lit. c Swiss UWG, *Compco Corp. v. Day-Brite Lighting, Inc.*, 376 U.S. 234 (1964). To the extent that work products are misappropriated that have been entrusted to another party, e.g. during a pitching phase prior to obtaining commissioned work, the unauthorized use of work results is partly sanctionable under unfair competition law (e.g. § 4 para. 3 lit. b and c German UWG, Art. 5 lit. a and b Swiss UWG, *International News Service v. Associated Press*, 248 U.S. 215 (1918)).

424 See THOUVENIN/BERGER, chapter 6/4.3, 2; RAUBER (1998), 68 f.

425 See THOUVENIN/BERGER, chapter 6/4.3, 2 f.; RAUBER (1998), 73 ff.

426 *Compco Corp. v. Day-Brite Lighting, Inc.*, 376 U.S. 234 (1964).

427 Decision of the OGer of Zurich of June 4, 1996, S2/U/SB960 165 – *Adressverwaltungsprogramm*; decision of the Court of the Canton of Zug of August 30, 1988, *Auto-CAD I*, published in SMI 1989, 58 ff.; decision of the Court of the Canton of Nidwalden of October 7, 1988, *Auto-CAD II*, published in SMI, 1989, 205 ff.; see also THOUVENIN (2018), 598, with further references.

428 THOUVENIN (2018), 598, referring to BGE 131 III 384 – *Suchspider* and decision of the Court of the Canton of Fribourg, published in sic!, 2017, 228 ff., 230 f.; See THOUVENIN/BERGER, chapter 6/4.3, 3.

429 For a detailed discussion on the relevance of unfair competition law for software, see: BOECKER, 110 f.; STRAUB (2011), N 522 ff.; CALAME (2007), 342 ff.

working products.[430] As the international legislation commonly allows copying and imitating if personal effort was invested, the protective scope for software is exhausted rather quickly.

3. Contract Law

229 A contract represents an agreement that creates obligations or claims between certain identifiable persons and is enforceable by law. It is based on mutual consent.[431] Contracts are frequently used to mutually regulate subject matter between several parties.

230 In a software contract "the issuer of a software obligates to produce, relinquish or maintain computer programs as well as all the necessary or useful documentation for the handling of the program to a user (for consideration)".[432] A software contract is not an alternative to the previous models but rather complements or combines some of them. The basis for a software contract is built on the (intellectual) property right of the right holder on the computer program. The contract itself has to observe the applicable contract law.

231 A software contract usually contains provisions that determine and specify a software's development and commercialization. It determines the usage scope of the program as well as the consequences of inappropriate use and governs who controls the associated commercial rights of the program. Through transparent and comprehensible rules, the collaboration and use of the software becomes foreseeable.[433] Software agreements can be subdivided into several different contracts, including standard software agreements, licence contracts, software development orders, service level agreements and so on. Depending on how close the customer is engaged in the development process, the characteristics of the final software product are predefined in the contract. Where standard software agreements are concluded – either through assignment, licences or service models – the computer program is commonly sold 'as is' and it is the software publisher who mainly fixes the terms of use.

---

430 See decision of the OLG Frankfurt of April 20, 1989, 6 U 213/88 – PAM-*Crash*, GRUR, 1989, 678 ff., 680; see also STRAUB (2011), N 29 and 30; BOECKER, 111 f.; THOUVENIN/BERGER, 6/4.3, 3 f.

431 See definition of the Legal Information Institute of the Cornell University Law School, "contract", available at < https://www.law.cornell.edu/wex/contract> (retrieved September 6, 2021).

432 Definition found in SLONGO, 7, brackets added.

433 WITTMER, 71.

The contracts usually also entail non-disclosure agreements for certain information regarding the software that is shared between the developer and the customer, and that the developer does not want to become public because it qualifies as a trade secret.

Most contract law, including the Swiss Code of Obligations, lacks substantive rules for software contracts; this type of contract is not legally regulated. As a consequence, a patchwork of different sets of law is applied. Software contracts are of additional importance in legislations with a weak IP software protection system; where developers cannot rely on their IP rights being enforced effectively, more contracts are concluded to ensure bilateral protection of the program. Market participants often use detailed contracts that settle every possible incident for software.[434,435] One disadvantage of software contracts is that they only have an effect *inter partes* and hence are only applicable between the concluding parties. Third parties are not bound by the agreed terms and consequently are not obliged to exhibit a particular behaviour towards the contracting parties. 232

C. Tabular Summary

The following table shows a brief summary of the described hybrid in legal software protection: 233

[434] FORSTMOSER, N 1.2 and 1.4; SLONGO, 21 ff; RUEESCH, 23 ff.; THOMANN (1992), 5; NIMMER/NIMMER (2014), N 27.

[435] See a good legal integration of agile and iterative development methods by means of contract law, in STRAUB (2015).

Legal Institution	Subject Matter
Intellectual Property Law	**Patent Law** New, non-obvious industrially applicable technical innovations, such as devices and procedures.E.g. implementation of algorithms and business methods, functions and features (disputed). **Copyright** An original and intellectual creation of art or literature. E.g. a completed software version, source code including its structure and organization, documentation, graphical user interface, look and feel (disputed) etc. **Industrial Design Rights** New and particular aesthetic or ornamental combinations of lines, contours, colours and surfaces contained in a useful product. E.g. visual elements, such as graphic user interfaces (disputed). **Trademark** A trademark is a sign or a combination of signs that is capable of distinguishing the goods or services of one company from those of other companies. E.g. designations of computer programs or ICT services. **Utility Models** A utility model (or petty patent) protects new, applicable and inventive devices from being copied or used without the consent of the right holder. They are particularly designed for 'minor inventions'. Excluded from protection are devices that solely rely on processing of information. E.g. hardware-related software inventions.
Unfair Competition Law	**Trade Secrets** Trade secrets are technical or commercial information that is considered as objectively secret and economically valuable for its secret-keeper and therefore worthy of protection from public discovery by competitors. They are protected if a third party obtains or shares them in an unrightful manner, even though the holder of the secret still has subjective intent to keep the information secret and after the secret keeper has taken reasonable measures to keep the information confidential.E.g. information about the logic, structure and ideas underlying a computer program, such as sensitive algorithms or hidden source code sections. **Taking Undue Advantage of Somebody Else's Achievements** If somebody offers imitations of somebody else's products or services and, either, has not exhibited a personal effort in the imitation process or the knowledge required to imitate was obtained in an unrighteous manner, the rules of fair competition are violated. E.g. one-to-one copies of a whole software product or of the source code.

Contract Law	Software contracts represent an agreed set of obligations between two or more parties to develop, assign or maintain a computer program on one side, and pay a reward for these services on the other side. E.g. service level agreements, licence contracts, software development orders, terms of use, usually combined with non-disclosure agreements for trade secrets.

II. Function of Intellectual Property Law

In order to understand why copyright and patent law are designed the way they are and how far their scopes of protection reach, the following section illustrates, in brief terms, the main functions and rationale behind intellectual property law and IP protection of computer programs in particular. 234

Intellectual goods are non-tangible. As neither their transfer nor their possession can be tied to a physical object, intellectual goods are difficult to regulate under classic property law. Instead, intellectual property law was established to assign the property on goods that lack a physical form, and to determine who obtains the associated control and commercial rights.[436] IP law has the function to steer and regulate according to a particular economic rationale. Today, all intellectual property rights are designed as government-acquiesced absolute exclusionary rights that are either granted to the creator of a work or invention, or a third party that has acquired the intellectual property from him/her by derivative means.[437] The exact composition of the protection depends on the respective property right. However, with partial reservation to trademark rights, intellectual property rights in general are based on three main common economic arguments: 235

First, intellectual property rights – and patents and copyrights in particular – belong to the person that has created the work or invention. It works as a *reward* for the one who was able to contribute something.[438] At the same time, it is assigned to its original owner. This is based on the principle in Natural Law 236

436 Thouvenin (2005), 255 f.; Hilty (2010), N 3; Hilti/Pedrazzini, 49 f.; Heinemann (2002), 21 f.

437 For more information, see Tom/Newberg, 171.

438 Beier, 234; Machlup, 377; Thouvenin (2005), 322; Kitch, 266; Dutfield/Suthersanen, 51 ff.; Thouvenin (2005), 287 f.

that a "human has an unconditional right on his [or her] own ideas".[439] The idea is legally assigned to its creator. The owner is therefore naturally free to decide on his or her property. This is a part of the creator's personality.[440]

237 Second, intellectual property rights possess specific functions. They aim to promote innovation and creativity and work as an *incentive* for a potential creator to invest in research and development by granting him or her exclusionary rights and artificial lead time.[441] The right holder obtains the possibility to enter a market alone for a restricted period of time in order to profit from his/her single dominant position, without a competitor competing in the same market.[442] Through the limited exclusionary (monopoly-like) right it is ensured that the supplier is able to market and hereby generate an exchange value for his/her non-physical, simultaneously usable good (that otherwise would be sensitive to free-riding) as a return to cover his/her past investments.[443] The creator is encouraged to develop and produce cultural goods and helpful inventions, which otherwise might remain undiscovered or unprotected in a free market.[444] Likewise, the U.S. Constitution states in Art. 1 sect. 8 para. 8 that the progress of science and useful arts can be *promoted* by securing for limited time to authors and inventors an exclusive right. The law therefore offers a legal infrastructure and an operational framework as a start-up support.[445]

238 Third, and most important from a social perspective, registry intellectual property rights, i.e. patents and designs, if granted, include a legal duty of the right holder *to reveal* his or her achievement with the public. By disclosing its

439 Machlup, 377; Commentary to the German UrhG (Loewenheim), introduction, N 9 f.

440 Dessemontet, 9 f.; Thouvenin (2005), 321 f.; Troller (1983), 68.

441 See as one of the first of his time: Kitch, 266 and 276 f.; for Europe: Thouvenin (2005), 288 f., 323 f. and 479 ff.; Machlup, 378 f.; Rossnagel, 69 f.; newer examples: Harison, 47 ff.; Samuelson et al., 2422; Marly, N 36 and 38; Commentary to the German UrhG (Loewenheim), introduction, N 13 f.; Hilti/Pedrazzini, 51; Dutfield/Suthersanen, 45; Heinemann (2002), 14 f. and 624 f.; Schwabach, 147 f.; Nimmer/Nimmer (2014), N 27-2; Koehler, 33; Nimmer/Nimmer (2016), N 1-106.3 f.; Dorr/Munch, 172; see also wording of Art. 1 sect. 8 of the U.S. Constitution which allows monopoly-like rights in order to promote the progress of science and useful arts, by securing for limited times to authors and inventors the exclusive right to their respective writing and discoveries.

442 Ullrich (1996), 555 f.; Heinemann (2002), 23 f.; Heinemann (2006), 705.

443 Fisher, 14; Scotchmer (2006), 34 ff.; Heinemann (2002), 15 ff. and 23 f.; Koehler, 32 f.

444 Reichman, 2434 f.; Dessemontet, 11; Rehbinder/Viganò, Art. 1 N 4.

445 For more information, see Rossnagel, 29.

contribution, society can profit from the developed knowledge and learn.[446] The knowledge exchange thereby increases public welfare, enhancing the public state of the art. This exclusionary right should correlate with its social value.[447]

239 It is the legislator's and judiciary power's responsibility to construct a fair balance between the far-reaching monopoly-like, absolute rights of a right holder, free competition and the needs of society. As a consequence, intellectual property law by design represents a compromise between diverging interests. Economists refer to this circumstance as a trade-off[448] between social benefits and private incentives. With the help of optimally designed intellectual property rights, the economic return of the right holder should "equal – but not exceed – the incentives it takes to induce both the successful and the unsuccessful creative efforts".[449] This means that IP rights should provide the creator with just enough incentive to develop and share his or her creation, without carrying it too far at society's expense.[450] As the creator does not have to rely on secrecy to protect his/her intellectual good from competitive actions, information can flow more freely and knowledge exchange is actively encouraged. The functionality and economic rationale behind intellectual property rights are consequently to reward the creator to develop and share his/her creation so that society can profit and learn from it.

240 There is still strong debate about how essential and to what extent the effects of intellectual property rights are positive for software in particular. Among the most common arguments against legal protection of computer programs under IP law is that computer programs can be protected with technical measures,[451] that their economic peculiarity makes protection by law unnecessary[452] and that intellectual property law rather harms social welfare and im-

446 Kitch, 275 and 278; Calame (2006), 9; U.S. Congress (1992), 184 f.; Boecker, 138; Marly, N 46 ff.; Hilti/Pedrazzini, 50 f.; Contu, 97; Dutfield/Suthersanen, 49 ff.; Koehler, 33; Commentary to the German UrhG (Loewenheim), introduction, N 17 and 19.

447 Scotchmer (2006), 98; Marly, N 48.

448 U.S. Congress (1992), 20 f. and 187.

449 Fisher, 14.

450 Decision of the German BGH of October 4, 1990, I ZR 139/89 – *Betriebssystem*, published in GRUR, 1991, 449 ff., 453; Boecker, 138, with further references.

451 Hoeren, 245.

452 See discussion in Boecker, 79 f., with further references.

pedes innovation.[453] Those in favour of intellectual property protection of computer programs – including the WIPO and the U.S. Congress – believe that modern society relies on information technology. Computer programs are therefore not only important for the ICT industry but also to other types of businesses that run on it. As society wants inventors and creators to share their knowledge of software, its use and development with the public, intellectual property protection is offered to the creators to foster knowledge transfer and at the same time express recognition of the merits provided.[454] In particular, the developers should have the possibility to regain the high investment in terms of time, money and labour that were required for the development of their creation. Intellectual property rights work as a long-term incentive for qualitative and quantitative increased discovery of technological achievements.[455]

241 Both camps, the ones in favour and those against intellectual property protection of computer programs, are encouraging a worldwide debate about the future legal integration of computer programs. Particularly vivid are the discussions on whether computer programs should be patentable. But apart from the patentability of computer programs, it should also be considered whether or not intellectual property law acts as an incentive and reward for the software developers to develop and share their creation with society and the users in particular. This question will be discussed in the findings.[456]

III. International Context

242 Intellectual property law represents one of the highest internationally standardized and unified fields of law of our time. Since the end of the nineteenth century, the international community has recognized that intellectual properties are suitable to manage international trade-related aspects. Therefore, as one of the first legal fields, intellectual property law was regulated on an in-

453 See discussion in U.S. Congress (1992), 23 and 135 f.; WIPO Standing Committee on the Law of Patents, 4 ff.; Mathematical Programming Society; Zirn, 149 f.; Melullis, 346 f.; Schwarz/Kruspig, 37 ff.; Walker, 54 f.

454 See for example: U.S. Congress (1990), 1.

455 Von Weizsaecker describes, on the basis of a theory by Eugen von Boehm-Bawerk, how this may cause a loss of efficiency in the short-term, which however would be tolerable to encourage more discoveries in the long-term (von Weizsaecker, 345 ff.).

456 For the findings regarding the question of whether computer programs should be eligible for patent law protection, see Chapter 5 Sections II.A.2. and V.B.2.d. For the findings regarding the function of software protection and its necessity, see Chapter 5 Section V.A.

ternational scale. This implies, on the one hand, that all the involved states were able to agree on the basic principles and minimum standards intellectual property rights should provide. On the other hand, the high density of international regulations causes the system of jurisdictional intellectual property law to be predetermined by international standards, leaving only little space for national regulations.

The following chapter presents an overview of the chronological development of the established international agreements that are relevant for the international interpretation of computer programs in copyright and patent law. With regard to the research question of this thesis, the following summary concentrates on the treaties and rules therein that are important to determine the scope of copyright and patent law or have a special focus on software. Interesting for the present thesis is how and to what degree the international community has firmly established computer program protection under copyright and patent law, and how much room for manoeuvre is still available for legislative adaptations. The international setting thus defines the legal framework in which this thesis can offer proposals for revision. 243

The subject matter under patent law and copyright protection is formulated abstractly in most treaties and therefore is quite flexible to newer interpretations of what may be regarded as protectable. On the other hand, certain other aspects, including the minimum terms of protection, are clearly defined and either have to be strictly complied with at national level, or have to be adapted through a tedious revision process within the international community. 244

A. Paris Convention for the Protection of Industrial Property

The Paris Convention for the Protection of Industrial Property ("Paris Convention"; SR 0.232.04) was concluded in 1883 and last revised in London in 1939. Patents, utility models, trademarks, industrial designs, trade names, geographic indications and further subjects in the context of industrial property in a wider sense all fall under the Paris Convention. 245

One of the treaty's main achievements is that it was the first agreement to internationally integrate patent law and thus provide rules in the field of patenting for many European countries that had not regulated this field before.[457] Today, the Paris Convention in its substance has been widely surrogated by the TRIPS Agreement and newer IP treaties. However, it offers a wider application 246

[457] See particularly Art. 1 of the Paris Convention.

field for national treatment of domestic issues.[458] Due to its early conclusion, the Paris Convention does not explicitly cover computer programs, describing its subject matter of IP rights in abstract terms. It can be assumed that in a modern interpretation computer programs will fall under the IP protection of the Paris Convention.

247 Of special interest for the present thesis are the rules on compulsory licensing in Art. 5 lit. A Paris Convention for patents. The signatory states are free to oblige the right holders to grant licences in case the exercise of exclusionary rights conferred by patents has an abusive effect (art. 5 lit. A para. 2 of the Paris Convention). The standards in the Paris Convention allow non-exclusive, non-transferable sub-licences to be available for four years after a patent has been filed or for three years after it has been granted.[459] The Paris Convention ensures that the patentee has the chance to justify his or her inaction, before a state-supported compulsory licence is established.

B. Revised Berne Convention for the Protection of Literary and Artistic Works

248 On September 9, 1886 originally ten states – including Switzerland, Great Britain, Italy, Germany, France and Spain – signed the Berne Convention for the Protection of Literary and Artistic Works (SR 0.231.15). Several revisions of the convention have taken place since, the most recent one in Paris on July 24, 1971 (amended on September 28, 1979). The treaty is commonly referred to as the Revised Berne Convention ("RBC"). As of October 2020, 179 countries had signed the Revised Berne Convention.[460] It is a multinational agreement in the field of copyright that aims to guarantee effective and uniform protection for the authors of literature and art.[461] For this purpose, the member states of the Convention have formed the Berne Union, providing rules for the organization of the Union (Art. 1 and 22 ff. RBC) as well as particular rules in the field of copyright that are mandatory for Union members (Art. 2 ff. RBC).

458 Ullrich (1995), 632 and 637 f.

459 See regulation in Art. 5 lit. A para. 4 of the Paris Convention. The rules are also applicable for utility models (Art. 5 lit. A para. 5 of the Paris Convention).

460 See official statistics with status as of October 1, 2020, available at <http://www.wipo.int/export/sites/www/treaties/en/documents/pdf/berne.pdf> (retrieved September 6, 2021).

461 See the preamble to the RBC.

Although some countries have refrained from signing the Revised Berne Convention and its scope widely overlaps with newer treaties, authors such as Schwabach state that it is still the most comprehensive treaty in the area of copyright, worldwide.[462] It is accepted as self-executing in most European countries[463] and can thus be invoked whenever a member state's law offers less protection than the Revised Berne Convention.[464] It is not regarded as self-executing in the United States.[465] 249

The Revised Berne Convention offers a set of substantive and procedural minimum standards. First of all, it defines in Art. 2 para. 1 that every literary and artistic work is subject to copyright protection. The article lists examples of accepted copyright goods as paintings, books and pictures but also plans and drafts in architectural and scientific contexts. Apart from the set of activities and rights that are reserved for the author of a work (Art. 8 ff. RBC), the Convention further determines which limitations and exceptions restrict the author's scope of action (Art. 10 and Art. 10bis RBC). According to Art. 7 RBC, copyrightable works are protected for a minimum duration of 50 years after an author's death. Individual states are free to implement shorter terms of a minimum of 25 years of protection for photographic works and works of applied arts after creation. Lastly, the Convention governs that copyright protection is not bound to any formalities but instead emerges automatically as soon as the requirements for copyright are fulfilled (Art. 5 para. 2 RBC). 250

Software is not explicitly mentioned in the Revised Berne Convention. However, the definition in Art. 2 para. 1 governs that every literary and artistic work is included. The list it provides is assumed to be only examples, as the formulation "such as" implies a non-exhaustive enumeration. It is thus concluded that because computer programs are made available either in a literary or graphic form, software is also eligible for copyright protection under the Revised Berne Convention.[466] 251

462 Schwabach, 19 f.

463 See for example BGE 105 II 57 f.; BGE 30 II 577 f.; decision of the ECJ of December 7, 2006, C-306/05 – *SGAE v Rafael Hoteles*.

464 See Art. 3 para. 1 and 2 RBC and its implementation in Swiss case law in BGE 62 II 245.

465 See U.S. Senate Foreign Relations Committee.

466 See corresponding interpretation of the U.S. in: U.S. Congress (1992), 12, 15 and 104 ff., and Schwabach, 151, with further indications of dependencies between the TRIPS Agreement and the Berne Convention; see also later indication in Art. 9 and 10 TRIPS, according to which WTO members are obliged to respect prior regulations of the Patent Cooperation Treaty and the Revised Berne Convention (Art. 9 para. 1 TRIPS).

C. Universal Copyright Convention

252 The Universal Copyright Convention ("UCC"; 0.231.01) was concluded on September 9, 1952 in Geneva and revised in Paris in 1971. The international community's intention, especially with the Paris version, was to offer a universal copyright that was acceptable for every nation and ensured the copyright protection of literary, scientific and artistic works around the world.[467] By offering contextualized international protection, the distribution of copyrighted works could be facilitated.[468] The aim was to find a common minimum regulation for all countries, so that countries that had previously refrained from participating in international agreements could now be integrated.[469] It particularly aimed to include the United States and several Latin American states that had not signed the Revised Berne Convention.[470]

253 Because the UCC is limited to a few basic principles, such as a minimum term of protection for copyrightable goods of 25 years[471] and the possibility for the signatory states to implement a special exemption for translations after seven years,[472]its scope and significance falls behind the Revised Berne Convention, particularly regarding its number of signatory states.[473] From a Swiss perspective, the UCC does not provide any particular improvement on the previous standards guaranteed under the Revised Berne Convention, which have already been implemented in the Swiss Copyright Act.[474] The UCC also contains several possibilities for states to restrict the comprehensive application of the treaty, again limiting the practical applicability of it in order to encourage prudent states.[475] Due to the early date the contract was concluded and revised, computer programs were not explicitly covered in the treaty. Although we can assume that the UCC is also applicable to software, its significance for IP protection in general is considered to be rather low.

---

467 Preamble of the UCC.

468 Preamble of the UCC.

469 Preamble of the UCC.

470 See explicit description in: DUBIN, 98.

471 Art. IV para. 2 UCC.

472 Art. V para. 2 UCC.

473 See discussions in GROSSENBACHER, 67 ff. and the current number of contracting party states of the UCC at <https://wipolex.wipo.int/en/treaties/parties/205> (retrieved September 6, 2021).

474 Same opinion in: RAUBER (1988), 56.

475 See for example Art. III UCC.

D. European Patent Convention

The European Patent Convention ("EPC"; SR 0.232.142.2) is a multilateral treaty that was concluded in Munich on October 5, 1973, and revised inter alia in 2000 and 2007. It is the most important source of rules concerning international patenting within Europe as it provides organizational and procedural rules for the European Patent Organisation, consisting of the Patent Office and the Administrative Council,[476] as well as fundamental principles for the European Patent.[477] The European Patent Convention is not a statutory act of the European Union, although most signatory states are also members of the European Union. Switzerland, which is not a member of the European Union, has also signed the European Patent Convention. The treaty aims to provide a Europe-wide patent union in order to strengthen international cooperation in protecting inventions.[478] The legal framework for granting European patents offers a directly applicable harmonized procedure for the European Patent Office. 254

In Art. 52 para. 1 the European Patent Convention offers a legal definition of the term patent. According to this, "European patents shall be granted for any inventions, in all fields of technology provided that they are new, involve an inventive step and are susceptible of industrial application". 255

Of particular interest for this thesis are the rules in Art. 52 para. 2 EPC, according to which mathematical methods, business methods, presentation of information and also computer programs are not eligible for the European patent. It is still unclear how this rule should be interpreted in practice. The article, although intensely disputed during the revision phase in 2000, remains unaltered.[479] As Art. 52 of the European Patent Convention is of major importance for the patentability of computer programs within Europe, it will be discussed in more detail later.[480] 256

476 See Art. 4 f. and 5 ff. EPC.

477 See Art. 52 ff. EPC

478 See preamble of the European Patent Convention.

479 See preamble to the Act Revising the Convention on the Grant of European Patents (European Patent Convention) of November 29, 2000.

480 See particularly N 273 ff., 280 ff., N 655 ff. for more information.

E. Agreement on Trade-Related Aspects of Intellectual Property Rights

257 The Agreement on Trade-Related Aspects of Intellectual Property Rights ("TRIPS Agreement"; attachment 1C to the Agreement Establishing the World Trade Company SR 0.632.20) stands alongside the Property Right Agreement on Tariffs and Trade ("GATT"; SR 0.632.21 and attachment 1A to the Agreement Establishing the World Trade Company SR 0.632.20) and the General Agreement on Trade in Services ("GATS"; attachment 1B to the Agreement Establishing the World Trade Company SR 0.632.20). The TRIPS Agreement was established in the Uruguay Round in 1994,[481] nearly 48 years after the GATT was concluded, and came into effect on January 1, 1995. It aims to foster international trade in the area of intellectual properties and offers, among other contexts, provisions on copyright and patent law. As the TRIPS Agreement is part of the WTO's conceptual framework, all 164 members of the WTO are bound by the TRIPS Agreement.[482] According to the WTO, the TRIPS Agreement is the world's most 'comprehensive multilateral agreement on intellectual property'.[483]

258 The TRIPS Agreement actively integrates the Revised Berne Convention and the Paris Convention, meaning that WTO members are obliged to respect these prior regulations (Art. 9 para. 1 TRIPS).[484] The Paris Convention and Revised Berne Convention thus both remain valid and their principles have become applicable also for member states of the WTO that have not signed the two treaties.

259 Among other rules, the TRIPS Agreement provides rules for the enforcement of intellectual property rights, in Part III, and measures for informal and procedural dispute resolution, in Parts IV and V. Of particular interest for the present thesis are the special standards for the scope, use and availability of copyright and patent law, in Part II: With regard to copyright protection, it states in Art. 10 para. 1 that computer programs are also eligible for protection as liter-

481 It constitutes Annex 1C to the Agreement Establishing the WTO.

482 Art. 1 para. 3 TRIPS Agreement; see current list of member states of the WTO at <https://www.wto.org/english/thewto_e/whatis_e/tif_e/org6_e.htm> (retrieved September 6, 2021).

483 For more information of the WTO on the TRIPS Agreement, see <https://www.wto.org/english/tratop_e/trips_e/trips_e.htm> (retrieved September 6, 2021).

484 This is true but Art. 6bis RBC on the moral rights of the author was disputed by the United States and is missing from the TRIPS Agreement. See for further information Schwabach, 151 and 237.

ary works in their source code or machine code form. It further states in Art. 9 para. 2 that the expression and not the idea of a work is protected under the TRIPS Agreement's copyright. Also of interest is the rule in Art. 12 TRIPS Agreement, under which "*whenever the term of protection of a work, other than a photographic work or a work of applied art, is calculated on a basis other than the life of a natural person, such term shall be no less than 50 years*". Similarly to Art. 7 RBC, it is thus possible for the member states to determine a shorter term of protection for photographs and works of the applied arts. At the same time, the TRIPS Agreement enables the term of protection to be tied to other points than the death of a work's author.

For patents, the TRIPS Agreement governs in Art. 27 para. 1 that "patents shall be available for any invention, whether products or processes, in all fields of technology, provided that they are new, involve an inventive step and are capable of industrial application." The treaty's subject matter for patent law thus involves all technical inventions that fulfil the protection requirements. It emphasizes that "patents shall be available and patent rights enjoyable without discrimination as to the place of invention, the field of technology and whether products are imported or locally produced". The significance of this rule is greatly disputed as many legislations exclude software as a whole, or particular components of computer programs from patent protection.[485] According to Art. 33 TRIPS Agreement, patents have to be protected for a minimum of 20 years in the member states' legislation. The treaty, in Art. 31 and Art. 31bis, further provides rules for a system of compulsory licensing in patent law. All in all, the TRIPS Agreement determines additional standards to the Paris Convention and the Revised Berne Convention regarding the provided scope of protection and clarifies some newer trends in international legislation. 260

F. WIPO Copyright Treaty

The WIPO Copyright Treaty ("WCT"; SR 0.231.151) was concluded as a special agreement under the Berne Convention on December 20, 1996 in Geneva. It builds on the status quo of the Revised Berne Convention, which is mandatory for all contracting parties.[486] In its preamble it emphasizes that the parties intend to maintain effective and uniform copyright protection by introducing new rules that summarize the current international understanding. It hereby aims to enable adequate solutions to questions raised by new economic, social, 261

485 See the protection scope of patents in N 272 ff. and N 652 ff.

486 See Art. 1 WCT.

cultural and technological developments and support, in particular the progress of information and communication technologies. Switzerland, the United States, Germany and France, along with 108 other states, have signed the WIPO Copyright Treaty over the last two decades.[487]

262 The WIPO Copyright Treaty explicitly determines that computer programs (Art. 4 WCT) and databases (Art. 5 WCT) are eligible for copyright protection. Computer programs are, in accordance with Art. 2 para. 1 of the RBC, protected as literary works, regardless of their mode or form of expression. This means, positively formulated, that computer programs can be equated to other forms of literary work and, negatively, that computer programs are restricted to their literary form. The term 'databases', on the other hand, is defined neatly and is delimited from other forms of data or information material. In Articles 6 to 11, the WIPO Copyright Treaty offers a modern designation of a copyright holder's commercial rights and legally integrates technological measures to protect programs, electronic rights, management tools,[488] and trends in software commercialization such as the rental of computer programs. The WIPO Copyright Treaty's significance for software-related protection is immense. It built the basis for the European Copyright Directive, the European E-Commerce Directive in Europe and the Digital Millennium Act in the United States.

IV. Patent Law

263 This section covers the relevant basics concerning the protective scope for software protection in patent law to understand the subsequent findings of the interview series as well as for their later discussion. For a full overview, see the extensive specialist literature on patent law and software. In a first section, it is briefly explained what a patent is and which rights are related to it. In a second, more extensive, section, the scope of patent law as presently interpreted is elaborated. There is a look at what the different jurisdictions impute on the subject of patent protection and a discussion of the requirements to obtain a formal patent. In a further section, information is given on the beginning and end of patent protection from a time perspective by discussing the registration process and maximum terms of protection. This thesis will not cover the various exemptions to the patent holder's exclusionary rights, the

487 See current list of signatory states at <http://www.wipo.int/treaties/en/ShowResults.jsp?lang=en&treaty_id=16> (retrieved September 6, 2021).

488 The circumvention of these technical protective measures is sanctioned by copyright law.

legal barriers or limitations, as the focus of this work remains the original protection scope of computer programs under patent law (and copyright).[489] On this matter, I therefore refer to the works of other authors.[490]

In contrast to the copyrightability of software that, on the whole, is unchallenged in Europe and the United States, the patentability of computer programs is widely disputed. While the Americans Cohen and Lemley in their paper called the issue of patenting software a "matter for history books"[491], as it is undoubted and firmly established in the United States, in Europe the eligibility of software for patents remains unclear. Still, many companies seek patent protection for computer programs, as it offers an interesting alternative to copyright protection. In the following, the terms software patent and computer program patent are used as idioms. 264

A. Patent as an Intellectual Property Right

A patent represents an exclusionary right that is granted by a sovereign power, either the state or a supranational authority, to an inventor in exchange for publicly disclosing his or her invention. 265

It provides the right holder with an absolute right. To the extent that no exemption or statutory limitation is given to the rights conferred, the patent holder possesses a legally protected right to forbid anybody else in the patent's territory to use his or her invention in any commercial way, regardless of the way in which the invention is expressed.[492] How far this prohibition right goes, depends on the nature of the patented object: 266

- Where a *device* is patented, the patent gives the patent holder the right to prevent third parties from making, using, offering, selling or importing the product;[493]

489 How legal barriers and exemptions may restrict the rights related to an affirmed patent is subordinate to the present research question, because these exemptions do not have an effect on the determination of the protection scope, but rather on the execution of the obtained rights.

490 See particularly Straub (2011), **various** quotations; Calame (2006), 459 ff.; Woestehoff for the U.S. Doctrine of Exhaustion.

491 Cohen/Lemley, 4.

492 See Art. 28 TRIPS Agreement.

493 Art. 28 para. 1 lit. a TRIPS Agrement, 35 U.S. Code § 271 lit. a and c, Art. 8 Swiss PatG.

– where a *process* is patented, the patent allows the patent holder to prevent third parties from using the process and (at least in the case of procedural claims) from using, offering, selling or importing products that were directly obtained by applying this process without consent.[494]

267 With the patent's associated rights, the patent holder has the possibility to establish on the market for a limited period of time without a competitor being able to interfere or apply the same, a similar or an equivalent invention. The patent holder receives an artificial lead time and thereby profits from a head start.[495] But the right holder can only prevent others from carrying out activities within the scope of the patent he or she was granted in the country as described in the patent claim(s). A crucial factor is the objective meaning of a claim, how it is to be understood by an expert in the field.[496] Further, the patent holder obtains a right of defence and the right to exclude others, but it does not necessarily involve them using the invention commercially.[497] The patentee also has the right to assign the patent or to transfer it by succession or to conclude licensing contracts.[498]

268 Once a patent is granted, patent law provides a strong degree and comparatively wide scope of protection. In most patent systems not only similarities to the literal scope of a patent claim are prohibited, but also what can be subsumed as equivalent under the patent scope (so-called *equivalence doctrine*).[499] According to the practice of the Swiss Federal Patent Court, equivalence has to be assumed if the replaced features have the same objective function (same effect), if the replaced features and their same objective function are obvious to a person having ordinary skills in the art on the basis of the teaching of the patent (accessibility), and if after reading the wording of the claim in light of the description, a person having ordinary skills in the art would consider the replaced features as a solution of equal value (equal value).[500] The equivalence doctrine was established with the same essential principles in the United

494 Art. 28 para. 1 lit. b TRIPS Agrement, 35 U.S. Code § 271 lit. a and c, Art. 8 and 8a Swiss PatG.

495 U.S. Congress (1990), 12.

496 BGE 122 III 81, c. 4.

497 See for example: *Herman v. Youngstown Car Mfg. Co.*, 91 F. 579 (6th Cir. 1911); Marbach/Ducrey/Wild, N 176.

498 Art. 28 para. 2 TRIPS Agreement.

499 See the Protocol on the Interpretation of Article 69 EPC of October 5, 1973 as revised by the Act revising the EPC of November 29, 2000, for the member states of the European Patent Organisation.

500 Decision of the Swiss BPatGer of March 21, 2013, S2013_001, c. 17.2.

States, Germany and in June 2017 also in the United Kingdom.[501] A third-party invention is not allowed to approximate to the invention in a way such that its substantial parts are the same, particularly similar or equivalent. As it is not only the final implementation that is protected by patent law, but rather the approach or idea as a whole, the effect is much greater and the scope of protection broader than for example in copyright.[502] An invention therefore needs to have a substantial distance from the patented solution in order not to infringe its rights.[503] An infringement occurs if the exclusionary right of the patent holder is violated. All activities that try to copy or imitate the patented invention, including aiding and abetting, are covered therein. Examples of infringing acts can be found in Art. 28 TRIPS Agreement. They particularly involve the use, the offering for sale, or the selling or importing of products and processes. The patent holder also has the right to demand information or to mark his/her products with a sign, indicating that it is patented. The patent statutories usually only contain non-exhaustive examples of the patentee's wide set of rights, limited by certain enumerated specific legal claims the patent does not entail.

The exclusionary right of the patent holder can be restricted by a legislator through statutory exceptions. These legal barriers and exemptions try to limit the potentially extensive effects of intellectual property rights in order to maintain a balance between various interests, and for example follow political reasons or pursue efficiency or social opinion.[504] 269

501 *Warner-Jenkinson Co., Inc. v. Hilton Davis Chemical Co.*, 520 U.S. 17 (1997); *Festo Corp. v. Shoketsu Kinzoku Kogyo Kabushiki Co.*, 535 U.S. 722 (2002); decision of the German BGH of March 12, 2002, X ZR 43/01 – *Schneidmesser II*, published in GRUR, 2002, 511 ff.; *Actavis UK Limited & Ors v Eli Lilly and Company*, UKSC decision of July 12, 2017 (UKSC 48).

502 See also Calame (2006), 659; Calame (2007), 339; Hilti/Pedrazzini, 158 f.; Staempfli Commentary to the Swiss PatG/EPC, Art. 1, N 57; Troller (1985), 619; Dutfield/Suthersanen, 123 f.; Samuelson (2017b), 1498 f.

503 This is part of the so-called "Doctrine of Equivalents", see Schwabach, 95.

504 In view of the focus of this thesis, this topic will not be further discussed in more detail. Instead, I refer to the comprehensive technical literature on the topic, especially Straub (2011).

B. Patent Scope

270 Patents are granted for a technical invention, provided that it represents an eligible subject matter, is novel and its finding was not obvious when it was made, the invention is industrially applicable or useable and the patentee is able to disclose his/her creation in sufficient detail.[505]

271 In this context, the patent scope refers to the borders and legal framework around this intellectual property right. As patents represent a government-supported monopoly-like right, the law defines what is regarded as a patentable subject, what requirements have to be fulfilled and what formalities are required. At the same time, the scope tells competitive legal entities which margins have to be overcome in order to place a rival good on the market without interfering with the patented good or service of a third party.[506]

1. Subject Matter

272 This section covers the subject matter of patents in order to determine what can be protected with this legal institution. As this problem is particularly complex for inventions in the field of software engineering, the different interpretations of Europe, Switzerland and the U.S. are summarized and illustrated.

273 The subjects of patent law protection are inventions in all fields of technology.[507] According to the Swiss Federal Institute for Intellectual Property, an invention from a legal perspective represents *a solution for a technical prob-*

505 See Art. 52 para. 1 EPC, § 1 para. 1 German PatG, Art. 1 para. 1 and 2 Swiss PatG, see also 35 U.S. Code § 101 ff.

506 HARISON, 15; U.S. CONGRESS (1992), 87.

507 Art. 52 para. 1 EPC, § 1 para. 1 German PatG, Art. 1 para. 1 and 2 Swiss PatG, 35 U.S. Code § 101.

lem.[508,509] The inventor sees a need, for which they seek a resolution with appropriate technical means.[510] Patent law therefore involves the protection of technical and engineering solutions as well as of substantiated ideas.[511]

Possible inventions – and therefore eligible subjects – can be *categorized* into procedures, products and applications:[512] 274

- *Claims for procedures* refer to manufacturing processes or a working method. The patent covers any application of the patented process, including carrying out the measures provided for in the patent claim or imitating them. It also covers the products directly resulting from the process.
- *Product, apparatus or composition claims* involve a corporeal object with a particular composition or form, mostly used as a device in a particular context. They enjoy absolute protection and any manufactured imitation of the product in question is prohibited, irrespective of the production method.
- An *application claim* is a process, art or method that includes a new use of a known teaching or object for a process, machine, manufacture, composition or material that is applied in a new context. Again, the patent forbids applying or imitating the invention.

---

508 See information of the Swiss Federal Institute for Intellectual Property at <https://www.ige.ch/de/uebersicht-geistiges-eigentum/die-schutzrechte-im-ueberblick/patentschutz> (retrieved September 6, 2021).

509 This is opposed to discoveries that refer to something already present in nature; see STRAUB (2011), N 433; see also TROLLER (1983), 148.

510 TROLLER (1983), 157.

511 Decision of the EPO Board of Appeal of May 15, 1984 (EPO T 0109/82) – *Hearing aid/Bosch*; Staempfli Commentary to the Swiss PatG/EPC, Art. 1 N 60; HARISON, 168 f.; BRINER, 49 f.; TROLLER (1983), 149 f.

512 Art. 52 para. 1 lit. a-d Swiss PatG, § 9 German PatG, 35 U.S. Code § 100; MARBACH ET AL. and STRAUB, are among the few to offer a good circumscription of these otherwise merely defined terms and their protection scope: MARBACH/DUCREY/WILD, N 43 ff.; STRAUB (2011), N 439 and 468 ff.; see also the summary in the Manual of Patent Examining Procedure of the U.S. Patent and Trademark Office, sect. 2106.03; for full description, see also BERESFORD, N 4.37 ff., N 4.40 ff. and N 4.45 ff.; LEMLEY ET AL., 147 ff. However, this classification is increasingly being questioned. Instead, patents are being reclassified as product (in German: "Erzeugnis") and process claims. See Guidelines for Examination of the European Patent Office, Part F-IV-3.1, and Swiss Guidelines for the Substantive Evaluation of Patent Applications, 50 f.

275 Most European statutes, including the European Patent Convention, determine that patents may only be assigned for inventions of "a field of technology". An invention must therefore exhibit a minimum degree of *technicity*.[513] The U.S. patent law does not mention the technical character of an invention explicitly.[514] However, newer contributions suggest that the U.S. patent practice likewise requires a technical content in the patent subject matter.[515]

276 How the technicity criterion should be apprehended in practice, is mostly unsettled.[516] There is no statutory definition. Instead its interpretation is left to the competent registry authorities and examining courts. The German Federal Court of Justice has suggested that *every controllable act that works with the help of manageable natural forces to achieve a particular causal success* could be defined as a technical one.[517] According to this interpretation, a technical invention thus involves a *task* and a *solution or teaching* in the form of a *repeatable success or a rule* that applies and manages the *natural forces*. This definition also suits the Swiss understanding of the term.

277 The differentiation between technical and non-technical inventions is specified in Art. 52 para. 2 EPC, where the convention outlines particular examples of objects that are considered as non-technical and which hence do not fall under the patent subject matter. These inventions are excluded because they either do not involve teaching or do not apply natural forces. A similar practice is followed in the United States. Common examples of such non-technical inventions include: abstract ideas, laws of nature, natural phenomena, scientific theories, mathematical methods, purely aesthetic creations and rules for men-

513 See Art. 52 para. 1 EPC, § 1 para. 1 German PatG; the Swiss Patents Act does not explicitly ask for technicity in the subject matter. According to prevailing opinion, and in accorance with the European Patent Convention, the term invention implies the requirement for technical teaching. See also Art. 1 para. 2 Swiss PatG e contrario, BGE 97 I 423 c. 1 and MARBACH/DUCREY/WILD, N 23.

514 35 U.S. Code § 101 does not request a technical invention or similar. See also DORR/MUNCH, 184 f.

515 A comparative study report of the European, Japanese and U.S. Patent and Trademark Offices also states that the practice of the U.S. Patent and Trademark Office requires "a technical aspect as a criterion for a statutory subject matter" (Patent Offices Comparative Study, 28; the same is implied by the Manual of Patent Examining Procedure of the U.S. Patent and Trademark Office, sect. 2106; see also WICKIHALDER, 583 f.; WALKER, 5.

516 BOECKER, 199; see also MARLY, N 417; HILTI/PEDRAZZINI, 83; ZIRN, 173.

517 Decision of the German BGH of March 27, 1969, X ZB 15/67 – *Rote Taube*, published in BGHZ, vol. 52, 74 ff., BGer of July 31, 1996, 4A.12/1995 – *Hochdruckkraftwerk*, published in sic!, 1997, vol. 1.

tal acts.[518] Only if these exceptions are applied in an inventive, creative way and if they include teaching can they come under patentability.[519] Also disputed for eligibility under patent law are business methods, which, like computer programs in general, are currently eligible for patent protection in the United States but widely inadmissible in Europe.[520] Marly therefore refers to this differing praxis as the "*divide*" between Europe and the United States.[521]

There are also certain matters that are explicitly assigned to the public domain in Art. 53 EPC and are thereby defined as unpatentable. In Europe a common example is goods that are considered unpatentable because they go against the public policy or morality of a society, as outlined in Art. 53 lit. a EPC.[522] In the United States, exceptions to the subject matter can only be found in case law. 278

Of particular interest for this thesis is the exclusion of "*computer programs as such*" from the patentable subject matter in Art. 52 para. 2 lit. c EPC, due to a lack of technicity. Even after many years of practice, it is still widely unsettled how far this exclusion goes and exactly how it should be understood. The case law and practice of the responsible authorities in all the jurisdictions have shown how difficult it is to guarantee a linear and unitary interpretation of the patent law's subject matter, and the technicity criterion in particular. To clarify 279

518 See Art. 52 para. 2 EPC and numerous case law: *Diamond v. Diehr*, 450 U.S. 175 (1981); *Bilski v. Kappos* 561 U.S. 593 (2010); BGer of July 31, 1996, 4A.12/1995 – *Hochdruckkraftwerk*, published in sic!, 1997, vol. 1; BGE 114 II 82; BGE 120 II 312; decision of the German BGH of March 27, 1969, X ZB 15/67 – *Rote Taube*, published in BGHZ, vol. 52, 74 ff.; decision of the EPO Board of Appeal of January 27, 1988 (EPO T 0281/86) – *Preprothaumatin*.

519 See, for Europe, Art. 52 para. 3 EPC and, for the United States, *Parker v. Flook*, 437 U.S. 584 (1978), in which the invention was ruled to be non-eligible for patent law. However, the considerations in the decision helped to open the door for a new interpretation.

520 See the different interpretations of business methods under European patent law, in Switzerland and in the United States, in the following subdivisions: N 294 ff., N 301 f. and N 305. As Straub outlines correctly, business methods are not themselves technical in nature. In the past, attempts have been made to monopolize business ideas, which are themselves not eligible for patent protection, through implementing them in computer programs. Consequently, the patent offices are forced to throughly examine whether the inventive step lies in the area of technical or non-technical elements (see discussion in Straub [2011], N 450, with further references).

521 Marly, N 414; see also Schwabach, 42 f.; Walker, 27.

522 See Art. 53 lit. a EPC and following litterae.

the status quo of practice, the different interpretations of the technicity criterion under the three legislations – the European Patent Convention, the Swiss Patents Act and the U.S. Code – are illustrated:

a) Interpretation under the European Patent Convention

280 According to Art. 52 para. 2 lit. c EPC, computer programs are explicitly excluded from the invention term. However, it is widely disputed to what extent computer programs can nonetheless be patented under the European Patent Convention. The European Patent Office has endeavoured to create a transparent and comprehensible practice, but the subject is difficult to grasp as the characteristics of the applied computer programs vary greatly. In addition the practice of the European Patent Office and the EPC's interpretation by the national patent offices and courts have changed many times over the last two decades, which impedes a linear European practice in this matter.[523]

281 Although the European Patent Convention explicitly forbids the patenting of computer programs, the European practice – including that of the European Patent Office and its Appeal Court – has allowed the patenting of software-related inventions under certain conditions. A corresponding attitude was also expressed in the proposal for the (rejected) Computer Program Patent Directive[524], which interpreted Art. 52 para. 2 lit. c EPC in a way that only computer programs "as such" are excluded from patent protection, while software-related inventions that propose a technical contribution would have been eligible for patenting.[525] The 'exclusion of computer programs' thus has to be understood as belonging to the enumeration of subjects that alone – without an essential addition – cannot be regarded as patentable technical teaching.

282 What may be understood by the required 'essential addition' depends on the current practice of the European Patent Office. In the last two decades, it has followed several different approaches to determine the patentability of computer programs under the European Patent Convention:

---

523 See also discussion in Calame (2006), 651 f.; Calame (2007), 333 f.; Dutfield/Suthersanen, 116 f.; Zirn, 49 f.; Melullis, 349 f.; Thouvenin (2010), 808.

524 Proposal for a Directive of the European Parliament and Council on the Patentability of Computer-Implemented Inventions of February 20, 2002 (C 151 E/129).

525 Preamble sect. 7 of the Proposal for a Directive of the European Parliament and Council on the Patentability of Computer-Implemented Inventions of February 20, 2002 (C 151 E/129).

Under the *any hardware* approach in the 1980s, the courts requested a minimum physical mechanical component.[526] A computer program was only considered as part of an invention if it was implemented in any form of hardware. This approach was rooted in the differentiation between touchable technology with a physical component, and the 'abstract world of ideas and contents of consciousness'.[527] However, in the VICOM case in 1986, the court clarified that inventions that fulfil the common patent criteria including novelty and industrial applicability should not be excluded from patentability only because they run with the help of a computer program.[528] The court concluded that the previous *any-hardware*-practice was too restricting. 283

As a consequence, the *contribution* approach was implemented. Under this, the state of the art became critical, asking for a technical contribution to the known art in a field that was not explicitly excluded from patenting.[529] Computer programs were thus considered patentable provided that an additional contribution from a field eligible for patents was included.[530] In the test, the patent office or court first had to identify the claimed contribution of the innovation. Then it was evaluated on whether it fell solely within an excluded area or also within an allowed subject matter, and whether the contribution actually exhibited a technical nature.[531] The contribution approach was abolished in the decisions *Auktionsverfahren/Hitachi*[532] and *Estimating sales activity/Duns Licensing Associates*[533] after 2004.[534] 284

526 See decision of the EPO Board of Appeal of December 12, 1985 (EPO T 16/83) – *Franceries/Traffic Regulation*; decision of the German BGH of arch 27, 1969, X ZB 15/67 – *Rote Taube*, published in BGHZ, vol. 52, 74 ff.

527 Decision of the German BGH of arch 27, 1969, X ZB 15/67 – *Rote Taube*, published in BGHZ, vol. 52, 74 ff. 76 f.

528 Decision of the EPO Board of Appeal of July 15, 1986 (EPO T 208/84) – *Computer-related Invention*, c. 16.

529 Decision of the EPO Board of Appeal of March 16, 1989 (EPO T 52/85) – *Listing of semantically related linguistic expressions/IBM*

530 Decision of the EPO Board of Appeal of July 15, 1986 (EPO T 208/84) – *Computer-related Invention*, c. 16.

531 Astron Clinica Ltd v Comptroller General of Patents, Designs and Trade Marks, decision of the EWHC of January 25, 2008 (EWHC 85).

532 Decision of the EPO Board of Appeal of April 21, 2004 (EPO T 0258/03) – *Auction Method/HITACHI*.

533 Decision of the EPO Board of Appeal of November 15, 2006 (EPO T 0154/04) – *Estimating sales activity/DUNS LICENSING ASSOCIATES*.

534 See Opinion of the Enlarged EPO Board of Appeal of May 12, 2010 (G 0003/08), N 10.4 f.

285 Today, the European Patent Office accepts software-related inventions if a *technical effect* can be observed.[535] According to the explanation in *Computer program Product/IBM*[536], a computer program…

> "… claimed by itself is not excluded from patentability if the program, when running on a computer or loaded into a computer, brings about, or is capable of bringing about**, a technical effect which goes beyond the 'normal' physical interactions between the program (software) and the computer (hardware) on which it is run"**[537]

286 The European Patent Office thus requires a technical effect and contribution to a computer program that exceeds the usual processes in a machine. A computer program might be patentable if it solves a problem and offers a contribution to the previous art that goes beyond the conventional physical interaction. The mere transmission of a stimulus within a computer program does not alone fulfil the technical effect requirement.

287 It is also unclear whether the technical effect has to lie in the problem that is solved, or in the problem-solving procedure. In the 1980s, the European courts applied the *core theory*, where the core of an invention had to be technical and the problem-solving procedure had to show technical characteristics.[538] For a couple of years, an alternative practice of the European authorities and courts has been observed. The *overall-consideration-theory* accepts a combination of technical and non-technical features if overall the invention involves an inventive step and if an interaction between two types of components is occur-

535 Guidelines for Examination of the European Patent Office, Part G-II-3.6; opinion of the Enlarged EPO Board of Appeal of May 12, 2010 (G 0003/08), N 10.2.1

536 Decision of the EPO Board of Appeal of July 1, 1998 (EPO T 1173/97) – *Computer program Product/IBM*; Decision of the EPO Board of Appeal of February 4, 1999 (EPO T 0935/97) – *Computer program Product/IBM*.

537 Decision of the EPO Board of Appeal of July 1, 1998 (EPO T 1173/97) – *Computer program Product/IBM*, c. 13; same wording in Decision of the EPO Board of Appeal of February 4, 1999 (EPO T 0935/97) – *Computer program Product/IBM*, c. 9.2 and 9.4.

538 The core theory was first implemented in the decision of the German BGH of June 22, 1976, X ZB 23/74 – *Dienstprogramm*, published in BGHZ 67, 22 ff.; see also decision of the German BGH of January 20, 2009, X ZB 22/07, – *Equipment for selecting medical examination methods*, published in openJur, 2011, 3117 ff.; decision of the EPO Board of Appeal of May 21, 1987 (EPO T 26/86) – *X-Ray Apparatus/Koch & Sterzel*.

ring.[539] The technical effect consequently does not necessarily have to occur in the problem-solving process. Instead, an overall consideration of all the features in the software-related invention and their impact on the functionality of the procedure or device is used.

If a computer program is connected to a machine, plant or device, and monitors, controls or checks this object, patentability is mostly undisputed.[540] This applies in particular to increases in efficiency or performance achieved by computer programs, which are accomplished in the hardware area and are hence 'physically' traceable. 288

The main difficulty remains in evaluating *computer-implemented inventions*. According to the Examination Guidelines of the European Patent Office,[541] the term refers to inventions that are executed solely on computers, computer networks or other programmable devices and are partially realized by means of computer programs.[542] Simplified, computer-implemented inventions lack a hardware component. As their technical teaching and the application of natural forces consequently only appear within their digital technical surroundings, they become difficult to distinguish from unpatentable inventions under Art. 52 para. 2 lit. c EPC. A case-by-case evaluation is required to analyse whether the merely digital inventions exceed the general level of 'normal physical interactions'. If a program does not clearly define its exact field and effect of application, the invention is regarded as an unpatentable set of information that lacks a technical teaching.[543] The classic literal understanding of a computer program as coded instructions to a machine are likewise excluded from the patentable subject matter, as, according to the European practice, they solely represent a mental act of formulating instructions that falls within 289

539 Decision of the German BGH of January 20, 2009, X ZB 22/07 – *Equipment for selecting medical examination methods*, published in openJur, 2011, 3117 ff.; decision of the EPO Board of Appeal of May 21, 1987 (EPO T 26/86) – X-*Ray Apparatus/Koch & Sterzel*; decision of the EPO Board of Appeal of July 3, 1990 (EPO T 603/89) – *Marker/Beattie*.

540 See for example: Hilti/Pedrazzini, 196; Commentary to the EPC (Melullis), Art. 52 N 202.

541 These are internal guidelines which are binding on the applying authority but not on third parties such as an applicant.

542 Guidelines for Examination of the European Patent Office, Part G-II-3.6.

543 Staempfli Commentary to the Swiss PatG/EPC, Art. 1 N 13.

the exclusions.[544,545] Again, if an inventor wants to patent them, an additional technical effect has to be exhibited, making it very difficult to patent inventive improvements in the technical science of software engineering alone because these advances are likely to be considered as a simple improvement of a mental instruction.

290 The handling of computer-implemented innovations remains difficult. For this reason, the EPO Guidelines for Examination provide several examples of computer-implemented inventions that the EPO has confirmed exhibit a technical effect, and thus are regarded patentable under the European Patent Convention:[546]

1. a computer program being a part of an invention that is used to *control* and *steer* industrial processes or devices;
2. a computer program being a part of an invention that improves *data transfer*, *data security*, *data storage* or improves *other resources* involved;
3. a computer program being a part of an invention that improves the *controllability of computer programs*;
4. a computer program being a part of an invention that offers a *new field of application or use for a known device*; and
5. an inventive mathematical method, that can be considered as a *technical contribution with a further technical effect itself*, that is implemented in a computer program.

544 Decision of the EPO Board of Appeal of April 16, 1993 (EPO T 204/93) – *Computer program Product/IBM*; decision of the EPO Board of Appeal of May 31, 1994 (EPO T 0769/92) – *General purpose management system*; decision of the German BGH of February 4, 1992, X ZR 43/91 – *Tauchcomputer*, published in BGHZ 117, 144 ff.

545 This interpretation of a computer program has been rightly criticized by various authors. As Boecker notes, a computer program is not just a simple instruction; it causes data to be processed to a facility (Boecker, 231 f.; same opinion: Troller [1987], 282 ff.); some authors suppose that the formulation is rooted in the obscurity of the legislator about the nature of computer programs back in 1970 which, until today, could not be fully resolved (Pfeiffer, 584 f.).

546 See examples in: Guidelines for Examination of the European Patent Office, Part G-II-3.6; see also brief summary in Marbach/Ducrey/Wild, N 27, and Staempfli Commentary to the Swiss PatG/EPC, Art. 2 N 13 ff. and 42 ff.

Discovering a computer algorithm that carries out some procedure, therefore, is not enough to obtain a patent.[547] In order to fulfil the technicity requirement, 291

- the algorithm in a program has to have an effect on a hardware component (no. 1 above);
- the application of the algorithm in a program has to suggest a new and applicable or a greatly improved technical teaching for an old problem (no. 2/3 above);
- a known technical teaching is applied to solve a problem in a new revolutionary way (no. 4); or
- an entirely new teaching is formulated for a undiscovered problem in a computer program (no. 5).

The technical effect approach consequently factually excludes the patenting of inventions that are limited to mere *digitalization* of known products and processes.[548] The automation of a non-technical process does not represent a patentable invention, if no additionally novel, non-obvious technical teaching is entailed.[549] 292

The EPO's expression 'computer programs "as such"' suggests an understanding of computer programs as mere literary expressions of instructions. Instead, they should be regarded, as Frei explains it, as a cluster of technical (machine, process, technical effect) and non-technical features (e.g. an algorithm or function).[550] Patent law should not separate the two. Instead, they should be assessed and qualified individually so that their respective contributions as well as their role in their interaction can be evaluated for their patentability. It is true that computers are technical in the sense that they use natural forces such as electricity and magnetism, because otherwise they would not function or run. They also contain controlling instructions that, depending on their content, may or may not constitute technical teaching within the meaning of patent law. The latter has to be assessed individually to determine an invention's patentability.[551] In order to exceed the threshold of Art. 52 para. 2 lit. c EPC, and reach the level of a patentable computer program, it would however 293

547 Opinion of the Enlarged EPO Board of Appeal of May 12, 2010 (G 0003/08), N 13.5.

548 See also explicitly in European Guidelines for Examination of the European Patent Office, Part G-II-3.5.1; see also discussion in Marbach/Ducrey/Wild, N 28.

549 Staempfli Commentary to the Swiss PatG/EPC, Art. 1 N 52; Melullis, 351.

550 Frei, 102.

551 See Frei, 115.

suffice to demonstrate that the classic patenting requirements are fulfilled. As the English Court of Appeal outlined in *Aerotel Ltd v. Telco Holding Ltd, etc*,[552] European patents are granted for any invention which is capable of industrial application, is new and which involves an inventive step. With the *Comptroller's structured approach*, the court examined in four steps (1) how the claims of the applied patent could be properly construed, (2) what the actual contribution of the patent was according to the claim, (3) what should be regarded non-protectable because it falls solely within the excluded subject matter, and (4) whether the actual or alleged contribution is actually technical in nature.[553] By thoroughly examining the general conditions for patenting it should hence be possible to exclude methods and products that are not worthy of protection, and consequently also to ensure adequate protection against excessive impairment of economic development.[554] Following this interpretation, exhibiting that a computer program provides a contribution that exceeds the common interaction between a program and a machine does not represent an additional criterion: In classic patent law, *technicity* means that *a final instruction to act, a teaching* is contained. If the invention shows no actual teaching of its own, the mere combination of an idea with a computer does not make an invention technical in terms of the subject matter.[555] The European Patent Convention as well as the European Patent Offices and Courts here seem to currently base their interpretation on a contemporary social use of the technicity term rather than on a patent law one.[556] I therefore advocate returning to a legalistic interpretation of the technicity term. In particular, the description of the technical teaching in the patent specification needs to demonstrate

552 *Aerotel Ltd v. Telco Holding Ltd, etc.*, and *Neal William Macrossan's Application*, decision of the EWCA of October 27, 2006 (EWCA Civ 1371); see also commentary in Thouvenin (2007), particularly 676.

553 *Aerotel Ltd v. Telco Holding Ltd, etc.*, and *Neal William Macrossan's Application*, decision of the EWCA of October 27, 2006 (EWCA Civ 1371), particularly c. 7 and 40 ff.

554 See also discussion in Krasser, 961 ff.; Wickihalder, 586; WIPO Standing Committee on the Law of Patents, 22 ff.; Frei, 138 ff.

555 See particularly the decision of the EPO Board of appeal of September 26, 2002 (EPO T 641/00) – *Two identities/COMVIK*; decision of the EPO Board of Appeal of March 5, 1997 (EPO T 0333/95) – *Interactive animation /IBM*; decision of the German BGH of August 25, 2015, X ZR 110/13 – *Entsperrung von Touchscreens-Entsperrbild*, published in GRUR, 2015, 1184 ff.; decision of the German BPatG of July 9, 2013, 17 W(pat) 82/09.

556 See similar discussions in Hilti/Pedrazzini, 86 f., 197 f. and 200; Staempfli Commentary to the Swiss PatG/EPC, Art. 1 N 37 ff.; Zirn, 168; Melullis, 346 f. and 349.

what novel, non-obvious contribution the invention offers[557] and give guidance as to how to *apply the teaching* in order to solve a special problem.[558] While digitalization of an analogue process does change the way a technical signal is transferred,[559] it does not itself include any further teaching that was not already part of the state of the art, to be classed as novel. It only includes a way similar to what has already been revealed and is hence known or to be expectd. Similarly, the European Patent Office rightly denied a procedural claim that involved reading and storing identification data from a user card mechanically, without offering any novel and non-obvious technical teaching.[560] Only if a new teaching is included, either in the process or result, will it be deemed a patentable invention. By relying on the classic patenting criteria, we can hence determine that no items will become patentable that do not include the relevant technicity or technical teaching, such as non-technical inventions, abstract ideas or simple discoveries. But we can further ensure that the invention and contained teaching are disclosed satisfactorily. Finally, a computer program can be just as patentable as any other findings, if the common criteria are fulfilled and proven in the patent specification.[561] The legal exclusion of computer programs hence relies on an antiquated understanding of software from the 1970s and urgently needs to be adapted, which is also indicated to some extent by the fact that the European interpretation simply circumvents it. This regulation, although obsolete in that respect, leaves many questions unanswered.

---

557 Merges and Nelson summarize it as follows: "a specification of the invention, (...) describing the problem the inventor faced and the steps she took to solve it. It also provides a precise characterization of the 'bestmode' of solving the problem." The inventor formulates a set of claims which exhibit what the inventor believes to be the scope of the invention and the technological field it affects (MERGES/NELSON, 844). See also HILTY (2014), 290 f.

558 Decision of the German BGH of January 20, 2009, X ZB 22/07 – *Equipment for selecting medical examination methods*, published in openJur, 2011, 3117 ff.; decision of the German BGH of October 17, 2001, X ZB 16/00 – *Sucher fehlerhafter Zeichenketten*, published in GRUR, 2002, 143.

559 See also BORNHAUSER for further information.

560 Decision of the EPO Board of Appeal of March 19, 1992 (EPO T 854/90) – *Card Reader/IBM*, published in the official Journal of the EPO, 2013, 669 ff.

561 See similar reasoning in BERESFORD, N 2.24 ff. and N 3.71; Commentary to the EPC (Melullis), Art. 52 N 200 ff.; SCHUHMACHER, 150 ff. and 189; HILTY (2014), 290 f.; LANG, 117; BOECKER, 199; DUTFIELD/SUTHERSANEN, 116 f.; ZIRN, 52 and 176 f.; WELCH/MUELLER, 295 in particular; WALKER, 18; FREI, 138 ff. THOUVENIN/BERGER, 6/2.2., 2

294 *Business methods* refer to planned (repeatable) procedures in the business area designed to achieve a goal in an optimal way.[562] In computer programs, functions and features or functional elements of the look-and-feel may for example refer to business methods.[563] Although the wording in Art. 52 para. 2 lit. c EPC explicitly excludes business methods from the subject matter, in accordance with the interpretation outlined for the patenting of computer programs, they are, in practice, likewise eligible for patent protection, to the extent that the patent claims do not represent a mere abstract idea or description of a natural discovery but instead outline a technical teaching that can be implemented innovatively to solve a problem.[564] The teaching must add something to human knowledge and serve a particular purpose, suggesting an act to be completed with the method. This act must then either result in a new product, a new outcome, a new process, or a new way of producing an old product or a result.[565] A typical case would be, for example, a working application patent. However, as Straub outlines, for this purpose, it would not suffice that simple technical instructions or functionalities are implemented in computer programs. Rather, it must be demonstrated that the inventive step lies in the field of technical elements. From the perspective of patent law, business methods are, like computer programs, not technical in nature.[566] Again, connecting them to a technical device does not turn the description of a simple method, which itself does not offer a technical teaching, into something tech-

562 Calame (2006), 653, with reference to Jaenich, 283.

563 See for example the decision of the EPO Board of Appeal of January 20, 1985 (EPO T 0605/93) – *Dai Nippon*, and the decision of the EPO Board of Appeal of March 5, 1997 (EPO T 0333/95) – *In teractive animation/IBM*, and further lists and comments in Beresford, 111-131 and 133-145; for critique: Hilti/Pedrazzini, 107 f.

564 See also the decision of the UK Patent Office regarding *Merrill Lynch's Application* (1989), published in RPC, 1989, 561 ff.; *Aerotel Ltd v. Telco Holding Ltd, etc.*, and *Neal William Macrossan's Application*, decision of the EWCA of October 27, 2006 (EWCA Civ 1371); see also discussion in European Guidelines for Examination of the European Patent Office, Part G-II-3.6.2.

565 *Reynolds v. Herbert Smith & Co., Ltd.*, decision of the EWHC of November 27, 1902, published in RPC, 1903, 123 ff.; decision of the UK Patent Appeal Tribunal of February 24, 1938, regarding *Fishburn's Application*, published in RPC, 1940, 245 ff.

566 Straub (2011), N 450, with further references; Jaenich, 486 and 487; Walker, 45; Staempfli Commentary to the Swiss PatG/EPC, Art. 1 N 29; Commentary to the EPC (Melullis), Art. 52 N 206 f.; Thouvenin/Berger, 6/2.3., 1; Schwarz/Kruspig, 100 f.; Calame (2006), 667; usually business ideas only represent abstract ideas or excluded discoveries from nature. See also the decision of the EPO Board of Appeal of September 8, 2000 (EPO T 931/95) – *Controlling Pension Benefits System/PBS PARTNERSHIP.*

nical.[567] As the court outlined in *Aerotel Ltd v. Telco Holding Ltd, etc.* for computer programs,[568] to evaluate business method patent applications we have to analyse whether the disclosed contribution is technical in nature. By thoroughly examining the general conditions for patenting it should be possible to exclude methods which are overly general and not worthy of protection, and consequently ensure adequate protection against excessive impairment of economic development.[569]

The same should be true for the patenting of *mathematical formulae* in the form of an algorithm. The abstractly formulated algorithm alone would not be able to leave the scope of Art. 52 para. 2 lit. a EPC, as it could (theoretically) also be executed in the mind. Its combination with a technical component, such as a computer, would not make it technical in terms of patent law.[570] But if a technical tool is applied to solve a technical problem, and this can be expressed in a teaching, the invention should be eligible for patent protection.[571] 295

The European interpretation of the patent law's subject matter remains disputable. As Thouvenin emphasizes, drawing a line between unpatentable computer programs 'as such' and patentable computer-implemented innovations remains difficult, maintaining great legal uncertainty.[572] However, the European patent offices and courts do allow software patents – contrary to the wording in Art. 52 para. 2 lit. c EPC – if a satisfactory description for the patent is provided which is able to demonstrate how a teaching is applied to solve a technical problem. Mathematical formulae, mental instructions and business methods are consequently eligible for patent protection when implemented in 296

567 See particularly the decision of the EPO Board of Appeal of September 26, 2002 (EPO T 641/00) – *Two identities/COMVIK*; decision of the German BGH of August 25, 2015, X ZR 110/13 *Entsperrung von Touchscreens-Entsperrbild*, GRUR, 2015, 1184 ff.; decision of the German Patent Court of July 9, 2013, 17 W(pat) 82/09; see also the explanation in European Guidelines for Examination of the European Patent Office, Part G-II-3.5.1 and 3.6.2.

568 *Aerotel Ltd v. Telco Holding Ltd, etc.*, and *Neal William Macrossan's Application*, decision of the EWCA of October 27, 2006 (EWCA Civ 1371); see also commentary in Thouvenin (2007), particularly 676.

569 See also discussion in Krasser, 961 ff.; Wickihalder, 586.

570 Commentary to the EPC (Melullis), Art. 52 N 206 f.

571 Decision of the EPO Board of Appeal of July 15, 1986 (EPO T 208/84) – *Computer-related Invention*, c. 14; Staempfli Commentary to the Swiss PatG/EPC, Art. 1 N 22 f. and 37.

572 Thouvenin (2010), 808 f.; Thouvenin (2007), 664; same conclusion: WIPO Standing Committee on the Law of Patents, 3 f.; Beresford, N 11.01 ff.; Calame (2006), 676 f.; Hilty/Geiger (2011), 167; Wiebe, 39 ff.; Zirn, 99; Samuelson (2012), 165 f.; Schuhmacher, 179.

a computer program, if it can be shown how the technical considerations are used to solve a problem in an innovative way, and if their effect goes beyond the normal interaction between the computer program and the machine.

b) Interpretation under the Swiss Patent Code

297 The Swiss Patent Code (Swiss Patent Code, "Swiss PatG"; SR 232.14) does not explicitly exclude computer programs from its subject matter, nor does it require a technical effect. Nevertheless, the Swiss courts and the patent examining authority, the Swiss Federal Institute for Intellectual Property, have adapted their practice to the European Patent Convention, as Switzerland is a member state of the European Patent Organisation. Consequently, it is to some degree bound by the EPO's interpretation.

298 Although not explicitly mentioned in the Swiss Patent Code, the Swiss Federal Institute for Intellectual Property also requires an additional technical effect in a software-related invention – apart from simply being a technical program – in order to become patentable. The technical character can either lie in the method of how a problem is solved, in the characteristics of a method used, in the functionality that results from applying the method to a problem or in the technical considerations needed to achieve an invention.[573] Instructions of the mind or spirit are excluded from patent protection, if they do not make use of natural forces.[574] Consequently, computer programs "as such" are unpatentable to the degree that they do not involve additional technicity.[575] On the other hand, computer programs, business methods and mathematical formulae may be part of a patentable invention, if the applicant is able to demonstrate and disclose what technical teaching it followed to solve a defined problem.[576]

299 The Swiss Federal Institute for Intellectual Property recognized the difficulty in distinguishing patentable software-related inventions from non-patentable

573 Swiss Guidelines for the Substantive Evaluation of Patent Applications, 16.

574 BGE 95 I 579 – "*Planungsverfahren*"; Swiss Guidelines for the Substantive Evaluation of Patent Applications, 16 f.

575 BGE 98 Ib 396 – "*Computerauswertung*""; Swiss Guidelines for the Substantive Evaluation of Patent Applications, 16.

576 Same conclusion in Frei, 114 ff. and 156 ff.; Staempfli Commentary to the Swiss PatG/EPC, Art. 1 N 37 ff.; Zirn, 168.

ones. In its landmark but non-binding guidelines, it provides indications and gives examples of cases where a computer-implemented invention may be considered as an eligible subject matter:[577]

- If a procedure possesses technical characteristics, apart from being a technical computer program, that work together or are connected with a computer program in order to fulfil a technical problem;
- if a computer program is implemented or connected to a device;
- if a computer program is used in a computer to solve a problem that relies on a technical teaching, including technical considerations and their implementation.[578]

Consequently, in the Swiss interpretation, a technical effect is shown if the invention is able to demonstrate technicity, apart from simply being a computer program. The expectations on the exhibited technical effect are higher, if the transfer and processing of a signal has to be evaluated. Fewer expectations are set on the technical effect, if manipulation of data is claimed.[579] A simple list of instructions in a programming language does not in itself fulfil the requirement of technicity.[580] It is imperative for every patent application involving computer programs that the applicant can demonstrate comprehensibly the technical teaching that is applied to solve the technical problem.[581] This means that the inventions have to be demonstrated in a way that the specialists of the field in question can understand. 300

Also computer programs that apply business methods or are merely based on a mathematical formula are in general not considered as technical and cannot thus be classed as inventions.[582] However, if they involve technical considerations, a technical teaching, or apply technical means to execute the inventive 301

577 Swiss Guidelines for the Substantive Evaluation of Patent Applications, 15 ff.

578 This is the case if a program is involved in technical processes for producing data, monitoring machines or steering processes. It mainly covers applications and process claims where the ICT system links the technology with specific controlling or steering actions. See also Swiss Guidelines for the Substantive Evaluation of Patent Applications, 19, with notes.

579 See practice described in: Swiss Guidelines for the Substantive Evaluation of Patent Applications, 19.

580 Swiss Guidelines for the Substantive Evaluation of Patent Applications, 15 f.

581 Swiss Guidelines for the Substantive Evaluation of Patent Applications, 19; Staempfli Commentary to the PatG/EPC, Art. 1 N 13; Thouvenin/Berger, 6/2.3., 1 f.

582 Swiss Guidelines for the Substantive Evaluation of Patent Applications, 20.

teaching, they might be considered to have reached the required technicity.[583] These methods therefore have to contain both technical and economic (or industrial) aspects. The overall view of the invention object is key.[584]

c) *Interpretation under the U.S. Code and Affiliated Case Law*

302 In the United States, patents are granted for technological advances that provide a practical solution to a specific problem.[585] The United States is famous for its particularly generous definition of patentable subject matter, after a report of the U.S. Senate in 1979 suggested that 'anything under the sun that is made by man' would be eligible for patent protection.[586] Consequently, although not explicitly mentioned in the U.S. Code, software is also eligible for protection under U.S. patent law, provided that the requested requirements are fulfilled.

303 In the early years of U.S. software case law, computer programs were widely regarded as mathematical ideas in the form of algorithms and thus unpatentable.[587] In *Diamond v. Diehr* in 1980, the U.S. Supreme Court for the first time suggested that a computer program that was able to control the execution of a physical process was patentable.[588] It took ten more years until a U.S. court accepted the combination of an inventive software mathematical formula (algorithm) with a trivial physical step as patentable.[589] The field was finally opened up for other software-related inventions in *State Street Bank v. Signature Financial Group* in 1998. In the same decision, the court specified that, for software inventions, the patentee further had to demonstrate what useful, tangible and specific results the application of the technical teaching could provide.[590] Following on from this, U.S. courts have emphasized that a

583 See instructions in Swiss Guidelines for the Substantive Evaluation of Patent Applications, 17 and 20; see also PMMBl 1975 I 30 ff.

584 See instructions in Swiss Guidelines for the Substantive Evaluation of Patent Applications, 17 and 20.

585 HARISON, 64; DUTFIELD/SUTHERSANEN, 118 f.

586 See U.S. Senate; this interpretation was later confirmed and adopted by the U.S. Supreme Court in: *Diamond v. Chakrabarty*, 447 U.S. 303 (1980).

587 See *Gottschalk v. Benson*, 409 U.S. 63 (1972).

588 *Diamond v. Diehr*, 450 U.S. 175 (1981).

589 *In re Alappat*, 33 F.3d 1526 (Fed. Cir. 1994); confirmed in *State Street Bank & Trust, Co. v. Signature Financial Group, Inc.*, 149 F.3d 1368 (Fed. Cir. 1998).

590 *State Street Bank & Trust, Co. v. Signature Financial Group, Inc.*, 149 F.3d 1368 (Fed. Cir. 1998), N 24 ff.; confirmed in AT&T Corp. v. Excel Communications, Inc., 172 F. 3d 1352 (Fed. Cir. 1999).

teaching implemented in an abstract program will not be patentable if it is not outlined how the potentially abstract invention has to be applied in an operation.[591] According to a later clarification by the U.S. Congress, software-related inventions have to involve a specific description of how and which functional steps are carried out by the system or which functions should be demonstrated with a particular computer program.[592] Mathematical algorithms in a computer program therefore become patentable if one is able to demonstrate that they involve a technical teaching that can be applied in a specific real-world context.[593] It thus became key for the patentability of an invention how specific a claim was, and its application was.

304 The U.S. courts have established various tests to examine whether a computer program falls under the patent law's subject matter. First, under the *machine-or-transformation test*[594] a process is eligible, if "it is tied to a particular machine or apparatus, or it transforms a particular article into a different state or thing".[595] Although this approach helped to isolate patentable from unpatentable software-related inventions, according to the Supreme Court's later opinion, the machine-or-transformation test was not able to sufficiently determine all patentable subjects but instead artificially narrowed the possible analysis results.[596] With a change in court practice and the expansion of the patent scope to every invention that was "specific" enough under the mentioned State Street Bank practice, a new test had to be established to evaluate the abstractness of a software-related invention. The court would determine whether the patent claims were formulated in too abstract a way, or were too generic or broad. In a second step, the court evaluated whether the applicant demonstrated comprehensibly how the invention was to be applied. Only if the method was described in detail and it was outlined how the problem would be

591 *Alice Corp. v. CLS Bank International*, 573 U.S. 208 (2014); *Edekka LLC v. 3Balls.com, Inc. et al.*, docket no. 2:2015cv00541 JRG (Eastern District Court Texas, 2015).

592 U.S. Congress (1992), 132; Walker, 7 f.

593 U.S. Congress (1992), 17.

594 *Gottschalk v. Benson*, 409 U.S. 63 (1972); *Diamond v. Diehr*, 450 U.S. 175 (1981); *Parker v. Flook*, 437 U.S. 584 (1978); further specified in: *Bilski v. Kappos* 561 U.S. 593 (2010).

595 *Bilski v. Kappos* 561 U.S. 593 (2010), 90; see also interpretation in Fusco and Schwabach, 24.

596 Supreme Court Opinion, 3 f.

solved, could a patent be granted.[597] In contrast to the European practice, the once permissive U.S. patent practice suddenly became stricter about allowing indefinite and non-novel patents.[598]

305 A similar practice can be observed for *business methods*, for example methods describing an ICT process. In contrast to the European Patent Convention, business methods are explicitly accepted by the U.S. patent system, if the requirements of novelty, applicability and technicity are fulfilled.[599] In view of the described former, rather generous, patenting practice of the U.S. Patent and Trademark Office, they are however rarely able to stand up in court and their claim is not sufficiently specific (indefinite claim) or the business method lacks a concrete mode of implementation. According to *CLS Bank International v. Alice Corp*, a business method has to exhibit more than a simple inventive concept.[600] It has to describe how it can be applied and to what degree this teaching is novel.[601] The description has to explain what particular machine is used or what result can be achieved by applying the business method. As one of the first and most famous ICT business methods, Amazon was granted a U.S. patent in 1999 for the *One-Click Solution*, a particular procedure for ordering and paying online. The patent was confirmed in 2001, when competitor Barnes and Noble implemented a similar online system on their website, which as a consequence got interdicted.[602] The very same patent was declined by the European Patent Office and its Board in 2011, stating that it would be too obvious and that a reduction in the number of necessary steps to order online would not constitute an inventive step.[603]

---

597 *Loyalty Conversion Systems Corporation v. American Airlines Inc*, docket no. 2:2013cv00655 (Eastern District Court of Texas, 2015); *Digitech Image Techs., LLC v. Electronics for Imaging, Inc.*, 758 F.3d 1344, 1348 (Fed. Cir. 2014); see also further discussion in Manual of Patent Examining Procedure of the U.S. Patent and Trademark Office, sect. 2161.01.

598 See same conclusion in Allison/Lemley /Schwartz, 1097 ff. and 1106 ff.; Lemley/Burk, 85 f.; Samuelson (2013), 23.

599 *State Street Bank & Trust, Co. v. Signature Financial Group, Inc.*, 149 F.3d 1368 (Fed. Cir. 1998); see also further information on interpretation and U.S. practice in Manual of Patent Examining Procedure of the U.S. Patent and Trademark Office, sect. 2106.04; see also comment in Schwabach, 23 f.; Lemley et al., 153, 154 and 156 f.

600 *Alice Corp. v. CLS Bank International*, 573 U.S. 208 (2014).

601 *Alice Corp. v. CLS Bank International*, 573 U.S. 208 (2014); Content Extraction and Transmission LLC v. Wells Fargo Bank, National Association, 776 F.3d 1343 (Fed. Cir. 2014); see also discussion in Walker, 8 f. and 32.

602 Amazon.com, Inc. v. Barnesandnoble.com, Inc. and Barnesandnoble.com, LLC, 239 F.3d. 1343 (Fed. Cir. 2001).

603 *1-Click/Amazon*, decision of the EPO Board of Appeal of January 27, 2011 (EPO T 1244/07).

Overall, the patentability of computer programs under U.S. patent law remains legally undisputed.[604] The focus of an examination instead lies on examining whether a technical teaching is provided and whether the patent applicant is able to demonstrate sufficiently how the teaching has to be applied. Computer programs consequently fall under the patent law's subject matter in the United States, if an invention is able to show an applicable technical teaching. 306

d) Conclusion: Are Computer Programs Patentable?

At first glance, the European and U.S. approaches for software-related patents could hardly be more different; the European Patent Convention in Art. 52 para. 2 lit. c still entirely excludes computer programs from its subject matter. Meanwhile, the U.S. Senate in 1980 declared that patent protection should be open to every invention under the sun created by man, and shortly after the U.S. Patent and Trademark Office and competent courts accepted computer program-related inventions as patentable inventions. 307

Nevertheless, the European Patent Office and European courts have established a practice that enables the patentability of software-related inventions under certain circumstances. Under the current approach, computer programs are eligible for patent protection, provided that they demonstrate a technical effect that goes beyond the normal interaction between a program and a computer. The U.S. practice, on the other hand, has become more stringent and in particular set higher expectations for the description of claimed inventions. The applicant has to specify how the invention will solve a particular problem and how this teaching will be applied. While the European patent system therefore sets particular expectations on what may be considered as a sufficiently technical subject matter, the U.S. one focuses on a certain minimum level of concreteness that the description of the invention has to exhibit in order to be considered a comprehensible technical teaching. 308

In effect, the U.S. interpretation does not differ so much from the European one; in both regions, patenting software-related inventions is possible with certain provisions.[605] 309

604 WIEBE, 79 ff.

605 See same conclusion in WIPO STANDING COMMITTEE ON THE LAW OF PATENTS, 2 f.

2. Protection Requirements

310 The protection requirements refer to the inventive threshold that has to be overcome in order to be granted a patent. Due to the high internationalization of the topic, the European Patent Convention and U.S. patent law have similar basic requirements. In order to be granted a patent in the United States, inventions have to be new and inventive as well as usefully applicable. These criteria correlate with the ones in Art. 52 para. 1 EPC, which state that a patent may be granted if a technology involves an inventive step, is new and is capable of industrial application.

a) Novelty

311 The provisions in Art. 54 EPC and § 101 and § 102 of 35 U.S. Code provide that an invention is considered as novel when it stands out from the state of the art. Hence, only progress of the present knowledge is patentable.

312 In order to evaluate whether an invention is novel, an examiner first has to determine the current state of the art in order to then verify in a second step whether the invention represents an improvement thereof. According to Art. 54 para. 2 and 3 EPC the state of the art consists of everything that is available to the public before the filing date of the patent application by means of a written or oral description, by use or in any other way.[606] A similar entitling is provided in the U.S. Code, further explicitly enumerating sold goods as part of the relevant prior art.[607] If by the time of the application, an expert of the field had the possibility to take notice of the claimed technical teaching, so that he or she had been able to apply the invention or conduct the inventive process, the invention would not be considered as novel.[608] The invention hence has to represent an enhancement to the prior state of the art and may not simply retain something that has been established but not yet patented. It is not required that the expert of the field has actually observed the teaching. According to von Bueren et al., it suffices that the possibility would have been there

606 See similarly Art. 7 para. 2 Swiss PatG.

607 See 35 U.S. Code § 102 lit. a para. 1.

608 *Diastereomers*, decision of the EPO Board of Appeal of February 9, 1982 (EPO T 12/81); decision of the German BGH of January 17, 1995, X ZB 15/93 – *Elektrische Steckverbindung*, published in GRUR, 1995, 330; BGer of July 24, 1991, published in SMI, 1993, 145; MARBACH/DUCREY/WILD, N 58; BRINER, 95 f.; Commentary to the PatG/EPC, Art. 1 N 73 ff.; Commentary to the EPC (Melullis), Art. 54 N 29 ff.; TROLLER (1983), 160 ff.; DUTFIELD/SUTHERSANEN, 120 f.

to take notice of it.[609] Nevertheless, the information has to be publicly accessible and not just subject to personal know-how or trade secrets that are not disclosed anywhere.

In practice, other patent specifications are usually consulted to determine the state of the art. In the specification of the application, it is not just the claims but also the descriptions, drawings and summaries that work as the basis to discover the state of the art.[610] 313

The state of the art has to be determined for every field and technical invention separately. It is a complex task to elaborate and define it. The term 'inventive activity' is a vague legal concept that involves discretion of the examiner and requires a judgement call.[611] Both the European Patent Convention and the U.S. Code aim to provide guidance by explaining to what degree a type of invention may be considered as novel and which disclosures may be considered as innoxious.[612] 314

In the case of computer programs, the novelty of a computer-related invention is particularly difficult to assess. On one hand, the market is quite dynamic and changes occur rapidly making it difficult to establish if anyone has yet described a teaching in a scientific paper or other type of publication. Most technical solutions are simply embedded in software products and services and then concealed. At the same time, as computer-related developments are still a relatively new field, it remains unclear what exactly may be regarded as an improvement of the prior art. The software development industry has been critical of the fact that too many commonly used, but not yet patented, inventions are monopolized. The evaluation of 'prior art' thus becomes more key with regard to software inventions. To determine prior art, the classic discovery media of specialist literature and oral descriptions at expert conferences may be extended to newer platforms, such as blogs, forums and libraries, where many technicians exchange their knowledge and discoveries online. 315

b) Non-Obviousness

Only if, regarding the state of the art, an invention is *non-obvious* to a hypothetical person skilled in this area can it be deemed an inventive step or 316

609 Marbach/Ducrey/Wild, N 58.

610 For Switzerland, see BGE 94 II 285; Staempfli Commentary to the PatG/EPC, Art. 7 N 17; Calame (2006), 408 ff.; Hilti/Pedrazzini, 132 f.

611 See Boecker, 219.

612 See Art. 54 and 55 EPC, 35 U.S. Code § 102.

progress.[613] This restriction ensures that the only inventions protected are those that offer an improvement from the perspective of a skilled examiner.[614] The improvement not only has to be novel, it also has to be to some extent surprising or unexpected. Indications of a non-obvious invention may for example include a long-standing but unsatisfied need to overcome a prejudice, an area of constant failure by others, a surprise for experts and the enrichment of technology.[615] At the same time, the teaching has to have a certain degree of significance for the technical field, outlining some inventive strength and exhibiting an inventive step.[616] If it is self-evident, the invention does not represent an inventive contribution and is not eligible for patent protection.

317 Relevant for the examination are pre-existing information and insights and the interpretation of the fictive or hypothetical '*average*' *or* '*ordinary*' *expert*.[617] The expert to be taken into account is one skilled in all technical fields covered by the doctrine of the patent at issue who possesses average technical knowledge or who can acquire such knowledge with the help of a member of staff.[618] The expected average knowledge level varies depending on the art in question.[619] As Heinrich highlights, the skilled person does however not need to be an expert in the technical field or a specialist with outstanding knowledge. They do not have to have an overview of the entire state of the art, but must have 'sound' knowledge and skills, a solid education and sufficient experience.[620]

318 The 'obviousness' restriction is further used to distinguish a patent from a trivial patent, that is particularly important in newer, still evolving sciences such

613 See Art. 56 EPC, 35 U.S. Code § 103.

614 See discussion of BGer of December 18, 2002, in Ingres, vol. 4/03, 5; *Graham v. John Deere Co.*, 383 U.S. 1 (1966).

615 Hilti/Pedrazzini, 138 ff.; see also Troller (1983), 171 ff.; Dorr/Munch, 188.

616 For Switzerland: BGE 85 II 131; BGer of April 1, 1969, published in SMI, 1971, 127 ff.; BGE 89 II 109.

617 BGE 120 II 312 c. 4b; *Graham v. John Deere Co.*, 383 U.S. 1 (1966); Staempfli Commentary to the PatG/EPC, Art. 7 N 4; see also discussion in: Briner, 95 f.; Hilti/Pedrazzini, 135 ff.; Commentary to the EPC (Melullis), Art. 54 N 44; Troller (1983), 163 and 167 f.; Thouvenin/Berger, 6/2.2., 4.

618 BGE 120 II 73, 73 f.; *Graham v. John Deere Co.*, 383 U.S. 1 (1966).

619 Dorr/Munch, 187.

620 Staempfli Commentary to the PatG/EPC, Art. 1 N 80.

as the computer sciences. The criterion further helps to balance patent law with antitrust law and make it more proportional with regard to potentially powerful exclusionary monopoly-like patent rights.[621]

The European Patent Office applies the *problem-and-solution* test in order to assess whether an invention is non-obvious and an inventive step is provided. It involves three major steps:[622] 319

- determining the "closest prior art";
- establishing the "objective technical problem" to be solved; and
- considering whether or not the claimed invention, starting from the closest prior art and the objective technical problem, would have been obvious to a skilled person.

With this approach it is easier to evaluate whether a skilled person considers the contribution of the invention as a modification. If a modification to prior art could have been achieved by the examiner, an inventive step may be assumed.[623] 320

The U.S. Patent and Trademark Office, on the other hand, follows the *functional approach* originally presented in *Graham v. John Deere Co.*[624] and confirmed in *KSR Int'l Co. v. Teleflex Inc.* in 2007[625]. According to this newer approach, there are seven indications that suggest the obviousness of an invention:[626] 321

- if prior art elements are combined according to known methods to yield predictable results;
- simple substitutions of one known element for another to obtain predictable results;
- use of known techniques to improve similar devices (methods, or products) in the same way;

621 See Boecker, 219, with further explanation; Commentary to the EPC (Melullis), Art. 54 N 6 ff.; Troller (1983), 189 ff.

622 Guidelines for Examination of the European Patent Office, Part G-VII-5.

623 Errat/Gowling, Strathy & Henderson, chapter 4.2; Hilti/Pedrazzini, 130 f.; Staempfli Commentary to the PatG/EPC, Art. 1 N 102; Commentary to the EPC (Melullis), Art. 54 N 129; Troller (1983), 159 f.

624 *Graham v. John Deere Co.*, 383 U.S. 1 (1966).

625 KSR Int'l Co. v. Teleflex Inc., 550 U.S. 398 (2007).

626 Examination Guidelines for Obviousness of the U.S. Patent and Trademark Office, no. 2143.

- applying a known technique to a known device (method, or product) ready for improvement to yield predictable results;
- choosing from a finite number of identified, predictable solutions with a reasonable expectation of success ("obvious to try")
- taking a known work in one field, making a small adaption based on design incentives or other market forces and using it in the same field or a different one, if the variations would be predictable for someone of ordinary skill in the art;
- using some teaching, suggestion, or motivation in the prior art that would have led a person of ordinary skill to modify the prior art reference or to combine prior art reference teachings to arrive at the claimed invention.

322 All of these indications are used as reference points in U.S. practice but are not regarded as a complete and exhaustive list. In order to avoid the patenting of trivial inventions, the U.S. Patent and Trademark Office also issued the *Examination Guidelines for Obviousness* in 2010 describing in more detail the current case practice and actual fields of application. Recently, a stricter standard has been applied in the United States, also scanning for similar basic concepts, regardless of how different those programs are in implementation.[627]

323 The minimum inventive step a software-related invention has to exhibit is again subject to great dispute. Of particular interest for this matter is the *1-Click-Solution* by Amazon already mentioned before. The European Patent Office had the chance to review the controversial computer-implemented invention of Amazon that had been evaluated as patent-eligible under U.S.[628] and Canadian law[629]. The Board of Appeal – like the European Patent Office before – dismissed the appeal regarding the rejected patent application of Amazon due to a lack of an inventive step combined with an unpatentable subject matter (*in casu*: business methods). The main claim of Amazon's 1-Click-Solution was to reduce the number of interactions involved in selecting items and the amount of sensitive information being sent over the Internet, which could be intercepted.[630] The European Patent Office and the Board of Appeals both ar-

627 Lemley/Burk, 85 f.

628 *Amazon.com, Inc. v. Barnesandnoble.com, Inc. and Barnesandnoble.com*, LLC, 239 F.3d. 1343 (Fed. Cir. 2001).

629 *Canada (Attorney General) v Amazon.com, Inc*, decision of the Canadian Federal Court of Appeal, A-435-10, 2011.

630 *1-Click/Amazon*, decision of the EPO Board of Appeal of January 27, 2011 (EPO T 1244/07), c. 2 f.

gued that the technical procedural teaching was, although not publicly known and common, already in parts recommended as a potential method in specialist literature.[631] The final implementation by Amazon did not involve a sufficient improvement that could overcome the inventive threshold. The detailed evaluation of both the European Patent Office and the Board of Appeals in the *1-Click-Solution-case* set high standards for future evaluations. It also showed that the European Patent Office takes the problem of granting trivial patents very seriously, dismissing an application it considered trivial and non-inventive, even though already accepted in other jurisdictions.

We may conclude that both approaches, the European and U.S., largely rely on a case-by-case evaluation, depending on the exact technology and the subject in question. In order to obtain a high quality evaluation, it helps if the expert has an extensive overview and detailed knowledge of the subject. Although the non-obviousness criterion may appear to be clear in theory, practice has shown that the examiner has to argue very carefully if they decide that an invention is obvious. The borderline between a legitimate 'reasonable' and a not protected 'obvious' teaching or solution is often narrow. It seems particularly difficult to evaluate the obviousness of a patent when an established or well-known technical teaching is transferred to another field of technology, as in digitalization processes. The expert, again, has to verify whether only a simple transfer was made and no further contribution was offered, or whether the prior teaching was expanded, enhanced or adapted. If the adaption represents a change that is insignificant or minor, from a skilled expert's perspective, no inventive step is acknowledged.[632] If, however, a further effort was made and something unexpected or additional was provided, a digitalization process could reach the threshold of 'non-obvious'. 324

c) *Industrial Applicability*

According to the European legislation, an invention has to be industrially applicable in order to be covered under patent law.[633] The U.S. Code utilizes a different wording, speaking of 'useable' inventions (utility),[634] but in practice applies the requirement in an almost identical manner. Using the term 'applic- 325

631 1-*Click/Amazon*, decision of the EPO Board of Appeal of January 27, 2011 (EPO T 1244/07), c. 20.

632 1-*Click/Amazon*, decision of the EPO Board of Appeal of January 27, 2011 (EPO T 1244/07), c. 23; see also Marbach/Ducrey/Wild, N 82 ff.

633 Art. 52 para. 1 EPC.

634 35 U.S. Code § 101.

able', the European legislator means that an invention not only needs to be novel but also has to be able to be applied in some way in practice. It thus cannot just be of theoretical value. The term 'industrially applicable' suggests that the invention has to be subject to an industrial field.[635] The European practice however accepts application possibilities in a commercial domain,[636] which also complies with the U.S. practice. The tendency is therefore for an understanding of the term as utility rather than applicability in the industry in a classical sense.[637] In order to be accepted as applicable, the invention has to be the result of a controllable activity. This means that it has to be repeatable and not just due to an accidental occurrence.[638] This ensures that the invention would potentially be open for (controlled) industrial use.

326 Under a newer ruling of the U.S. Supreme Court, the court has now reclassified the criterion of usability, saying that computer programs, mathematical formulae and business methods have to demonstrate whether and how they are applicable.[639] An abstract implementation does not fulfil this requirement. As Lemley explains, software patent lawyers increasingly write patent claims in broad or ambigious functional terms, claiming not a particular way of use, as in a machine or a particular set of steps of a procedure, but rather the goal itself.[640] This creates too broad a scope of protection for the right holder, enabling the strongest possible rights for patentees.[641] With the new U.S. practice of industrial applicability or utilization, the United States has put a stop to this broad approach particularly debated in the U.S. market.[642] It also synchronizes with the European approach, asking for sufficient disclosure which clearly and fully sets out the patent claim:[643] The instruction that the patentee includes has to reveal a method that is clear and applicable for an expert of the field,

635 Art. 57 EPC; see also Thouvenin (2006)

636 See Boecker, 216, with reference to § 1a para. 4 German PatG; Troller (1983), 201.

637 Dutfield/Suthersanen, 122 f.

638 Decision of the German BGH of February 12, 1987, X ZB 4/86 – *Tollwutvirus*, published in BGHZ 100, 67 ff., 71 f.

639 *Alice Corp. v. CLS Bank International*, 573 U.S. 208 (2014); *Edekka LLC v. 3Balls.com, Inc. et al.*, docket no. 2:2015cv00541 JRG (Eastern District Court Texas, 2015); see also 35 U.S. Code § 112 as well as practical procedures during patent examination according to the Manual of Patent Examining Procedure of the U.S. Patent and Trademark Office, sect. 2107.

640 Lemley (2017), 905.

641 Lemley (2017), 907.

642 See also statistical analysis in Allison/Lemley/Schwartz, 1104 ff.;

643 Art. 83 EPC.

outlining what is used and how this represents an inventive step.[644] An expert should be able to understand what he or she has to do, following the instructions one-by-one. The more abstract an instruction is, the less chance of its eligibility under this criterion. A patent description should therefore be specific enough, so that its application is comprehensible and capable of being used. The extent of the disclosure approach varies according to the nature of the invention as well as the role and complexity of the computer program needed to implement it.[645] The claim at least has to outline the functionality on which it is based and which it wants protected. It would not be necessary, however, to disclose the source code in which it was implemented.[646]

Compared with the previous situation, in which the industrial applicability was less important as it was easier to fulfil in a patent evaluation, the criterion of industrial applicability thus has seen a revival, becoming more relevant and important whenever computer programs are evaluated under patent law.[647] 327

3. Term of Protection

If appropriate subject matter has been shown and the requirements have been fulfilled, an invention is eligible for patent protection. The exact scope of protection, however, is dependent on a time component: when the protection starts and when it ends. The following section briefly summarizes the relevant steps in the patent registration process and explains when the term of patent protection ends. 328

The patent term refers to the period of time between when the legal effects of a patent begin and the moment when they expire. During this time, the disclosed know-how is protected and can be commercialized through licences and other forms. 329

644 See also the decision of the UK Patent Office regarding *Merrill Lynch's Application* (1989), published in RPC, 1989, 561 ff.; see also Opinions of the Lords of Appeal for Judgment in the cause *Synthon BV v. Smithkline Beecham plc* of October 20, 2005, commented in THOUVENIN (2006).

645 *Northern Telecom, Inc. v. Datapoint Corp.*, 908 F.2d 931 (Fed. Cir. 1990).

646 *Fonar v. General Electric*, 107 F.3d 1543 (Fed. Cir. 1997).

647 See discussions in: OSTERRIETH, 992; BRINER, 60 ff.; Staempfli Commentary on the PatG/EPC, Art. 1 N 171; Hilty (2014), 290 f.; ZIRN, 180; LEMLEY ET AL., 181 f.; see further information in Manual of Patent Examining Procedure of the U.S. Patent and Trademark Office, sect. 2161.01.

a) Starting Point of Protection: Registration

330 The starting point for protection is when the patent application is filed for registration. But only if a patent is granted, and registered, can the exclusionary rights associated with the patent be enforced. By submitting an application, the private parties decide what they believe is worthy of protection by disclosing their invention in a patent specification that will be examined by an expert.[648] With the filing of the application, the inventor for the first time has to define the claim and the exact subject of protection. The description has to disclose enough information so that the state of the art can be evaluated properly, and the examiner can verify whether the invention is commercially applicable.

331 Enclosed with the patent application are several documents, such as a formal registration request, a description of the invention, one or multiple claims, any drawings referred to in the description or in the claims, and a summary of the invention.[649] The patent can either be filed at a national office, for example the Swiss Federal Institute of Intellectual Property, or at a regional patent authority such as the European Patent Office for a European Patent or within the Patent Cooperation Treaty Union for an application under the PCT procedure.[650] The decision as to where a patent is filed depends on strategic considerations. Some companies want to profit from a longer priority period that is provided by some legislations, others prefer a shorter application procedure in some jurisdictions, while others may want to benefit from a full evaluation and testing of the patent requirement and the subject matter in some areas.

332 After the application is filed, the procedure usually evolves according to the following steps:[651] (1) A high-level examination upon filing; (2) a closer examination of formal requirements, including a check on whether all the necessary documents were submitted; (3) a voluntary order for research of the state of the art; (4) publication of the patent application; (5) if scheduled and required,

---

648 MERGES, 594 f; TROLLER (1983), 451 ff.

649 Art. 78 para. 1 lit. a-e EPC, Art. 49 para. 2 lit. a-e Swiss PatG, 35 U.S. Code § 111 para. 2 - § 115.

650 The Patent Cooperation Treaty (PCT; SR 0.232.141.1) is a special cooperation agreement that was concluded in Washington on June 19, 1970 in order to support coordination and cooperation in the field of patents within the international community. It provides a special application procedure that facilitates and accelerates the patent filing and registration procedure and makes it more efficient in all signatory countries.

651 Art. 90-98 EPC; for the United States, see 35 U.S. Code § 111-135.

a substantive examination of the claim; (6) if the patent requirements are fulfilled, the patent is granted and the registry entry is processed. This is the end of the patent application and examination procedure.

How extensively the substance of a patent application is examined (step 5) depends on the country in which the patent application is filed. In Switzerland, currently only the subject matter has to be assessed by the competent authority, the Swiss Federal Institute of Intellectual Property, but not the patent requirements as such.[652] In the European Patent Office and in the United States Patent and Trademark Office, a full substantive examination is made.[653] If a substantive examination of all the patent requirements is made, the voluntary research (step 3) is skipped, as it is integrated into the examination process. Whether or not a substantive examination of the patent takes place is widely reliant on the policy a country or treaty pursues. The European Patent Organisation and the United States prefer to offer a comprehensive, in-depth evaluation of the patent to increase the quality of the granted patents and reduce the number of trivial ones.[654] The Swiss patenting procedure, on the other hand, is much faster as no substantive examination takes place.[655] The downside of the accelerated process is that a patentee's property right has not been tested by the responsible authority and thus they do not know to what degree it is able to withstand a substantive complaint in court.[656] 333

Only applicants that have passed all the necessary examination steps, provided the required documentation and paid the fixed fees will be granted a patent and registered as 'patent granted' in the patent registry. Once the application 334

652 Art. 59 para. 4 Swiss PatG and Swiss Guidelines for the Substantive Evaluation of Patent Applications, 8 f.; a very recent preliminary draft for a revised Swiss Patents Act newly provides for a full voluntary substantive evaluation of the patent requirements in Switzerland, resulting in the striking of Art. 59 para. 4 Swiss PatG. However, the consultation process was still ongoing at the time of printing. For more information on the revision process, see <https://www.ige.ch/en/law-and-policy/national-ip-law/patent-law/patents-act-revision.html> (retrieved September 1, 2021).

653 European Guidelines for Examination of the European Patent Office, Part C-I-4; Manual of Patent Examining Procedure of the U.S. Patent and Trademark Office, sect. 904.03.

654 Troller (1983), 498 f.

655 Troller (1983), 496 and 498.

656 See also interesting analysis regarding patents and validity in court, in Allison/Lemley/Schwartz, 1124 ff.;

procedure is complete, the patentee may make use of the rights that are affiliated with the patent. At the same time, after publication anyone can file an opposition procedure to formally question the granting of the patent.[657]

335 The formal starting point of protection for the disclosed information therefore is when the application is recorded at the patent office, and the rights associated with the patent only start when the patent is granted. The patentee's absolute commercial and interdiction rules consequently can only be exercised when the patent is formally registered.

b) End of Protection

336 The patent protection ends when the maximum period of protection is reached. The patentee can thus not profit from an absolute right indefinitely. Patent law determines explicitly after what time the patented innovation falls into the public domain and can thus be used without further authorization by society.

337 The rights associated with a patent expire *twenty years* after the application was filed (the application date) under the European Patent Convention, the Swiss Patents Act and the U.S. Code.[658] The patent protection may also end before the maximum term of protection, in a case where:

- the annual patent fees are not paid, which is the most common cause;[659]
- the patentee retracts his/her patent right;[660]
- or the patent is declared invalid (revoked).[661]

The scope of the patent may also be limited without the patent protection ending. By revoking or restricting one claim or merging several patent claims, the patent scope is reduced and the patentee's rights may comprise a smaller dimension. Such later limitation of the scope of protection represents a partial waiver and has a retroactive effect on the whole term of protection (*ex tunc*).[662]

---

657 Art. 99 ff. EPC, 35 U.S. Code § 135.

658 Art. 63 para. 1 EPC, 35 U.S. Code § 154 lit. a para. 2.

659 Art. 51 EPC, Art. 15 para. 1 lit. b Swiss PatG, § 20 German PatG, 35 U.S. Code § 41 and § 154 lit. a para. 2.

660 Art. 105a EPC, Art. 15 para. 1 lit. a Swiss PatG.

661 Art. 105 ff. and Art. 138 EPC, Art. 26 para. 1 and Art. 28 Swiss PatG, § 21 f. and § 82 ff. German PatG.

662 Art. 105a EPC, § 21 para. 2 and 3 as well as 39 German PatG, Art. 24 f. and Art. 28a Swiss PatG and further description in MARBACH/DUCREY/WILD, N 227 ff.

Generally, therefore, a patent can be valid for a maximum of twenty years after the application was filed. The term of protection may, however, be shortened for various reasons, or the patent scope reduced. 338

V. Copyright

In contrast to the big disputes regarding the patentability of computer programs, applying copyright to computer programs represents a standard use case and is embedded in international contracts. This section provides a short introduction to the most important points considering software protection in copyright law, to enable an understanding of the findings of the interview series and the later discussion. For a full overview, please consult the extensive specialist literature on copyright and software. First, it is explained what copyright entails and, briefly, which commercial and moral rights are associated with it. Second, there is a closer look at what falls under the protection scope of copyright, including the subject matter, the requirements that have to be fulfilled to obtain copyright protection, and the start and end points of the term of protection. Similarly to the previous section on patents, this section will not fully discuss the rights granted in copyright law nor address the legal barriers or limitations that are available to restrict the rights of copyright holders. On this matter, I refer to the many works of other authors.[663] The focus of this thesis remains the protection scope of computer programs under the perspective of copyright and patent law. This introduction will therefore only explain the basics of the copyright scope, which are independent of how a copyright holder may or may not execute their legal rights afterwards. 339

A. Copyright as an Intellectual Property Right

Copyright is intended to cover creators and practising artists by granting protection to their creative works in art and literature.[664] As Charles A. Richard explained it back in 1976, copyright wants to "tribute the outstanding practitioners (...) whose (...) innovations are based on superb skill in the practice of 340

663 See particularly Straub (2011); Woestehoff for the U.S. First Sale Doctrine; Thomann (1998); Marbach/Ducrey/Wild.

664 Preamble and Art. 2 para. 1 RBC, Art. 1 para. 1 lit. a Swiss CopA, 17 U.S. Code § 102 lit. a, § 1 German UrhG.

their craft, combined with an acute insight into the underlying principles."[665] Copyright hence covers the work of creative and skilled people who share their creation with society.[666]

341 The author of a copyrighted work can only be the person who created the work. Copyright originally assigns the exclusive and absolute right of a creative work to him or her.[667] These rights can be subdivided into commercial and moral rights. The *commercial rights* in this context refer to the economic utilization of the work. The intention is to give the author the possibility to earn back his/her investment.[668] The most common commercial rights associated with copyright are the right to control the use of a copyrighted work;[669] the right to decide who can reproduce the work;[670] the right to distribute the work, or copies of it;[671] the right to distribute a work for rental;[672] and the author's right to perform the work, to display it and to make it available for the public.[673] The various commercialization models of purchasing, licensing and service are based on these available commercial rights and also enable an author to transfer or assign his or her rights to a derivative right holder. The *moral rights*, on the other hand, are a pecularity of copyright.[674] They reserve some particular rights to the author alone, in order to protect and support the personal contribution and connection of an author to their work. The most

665 Foreword by Charles Antony Richard in: DIJKSTRA, xi; see also discussion in HILTY (2010), N 36.

666 BARRELET/EGLOFF, Art. 1 N 4.

667 Art. 6 in conjunction with Art. 9 para. 1 and Art. 10 para. 1 Swiss CopA, 17 U.S. Code § 106 f., § 15 German UrhG; see also Art. 9 TRIPS Agreement in conjunction with Art. 9 para. 1 RBC.

668 See for example HILTY (2010), N 149.

669 Art. 10 para. 1 Swiss CopA, § 15 in conjunction with § 69c German UrhG, implicitly 17 U.S. Code § 106.

670 In terms of software this right widely focuses on the exclusive right to make legal copies of the protected program; Art. 9 para. 1 RBC, Art. 10 para. 2 lit a and b Swiss CopA, § 16 in conjunction with § 69c para. 1 German UrhG, 17 U.S. Code § 106 para. 1, Art. 2 Copyright Directive, Art. 4 para. 1 lit. a Computer Program Directive.

671 This includes the right to decide, in what form and where it is offered and how copies of the software will be distributed and sold; Art. 10 para. 2 lit. b Swiss CopA, § 17 in conjunction with § 69c para. 3 German UrhG, 17 U.S. Code § 106 para. 3; Art. 4 Copyright Directive, Art. 4 para. 1 lit. c Computer Program Directive.

672 Art. 10 para. 3 Swiss CopA, § 17 para. 3 in conjunction with § 69c para. 3 German UrhG, 17 U.S. Code § 106 para. 3, Art. 4 para. 2 Computer Program Directive.

673 Art. 10 para. 2 lit. c-f Swiss CopA, § 18 ff. in conjunction with § 69c para. 4 German UrhG, 17 U.S. Code § 106 para. 4-6, Art. 3 Copyright Directive.

674 See for example HILTY (2010), N 149; MARBACH/DUCREY/WILD, N 321.

relevant examples are the obligation to name the work's author, their right to decide when and in what form the work will be published for the first time and the right to protect the integrity of their work.[675] The existence of moral rights in software copyright is not undisputed, as computer programs represent a good of high economic and functional interests.[676] It is therefore a matter of untested discretion to what extent the moral rights can be executed and enforced in software engineering.

An author or right holder is partially restricted in the execution of his or her exclusive commercial and moral rights by the above-mentioned legal barriers. For more information on the various legal exemptions in copyright, please see the relevant specialist literature on copyright. 342

B. Copyright Scope

The scope refers to the borders and the legal framework that is implied by a copyright. As outlined above, copyright covers creative works or expressions in the field of literature and art. This represents the subject matter of copyright. To fall under the copyright's scope, a creative work has to be (1) original (or individual) and (2) intellectual in order to be protected.[677] 343

The following sections illustrate what can be protected under copyright law and what requirements a creation has to fulfil to be covered. Again, the focus will lie on the jurisdictions of the United States, Switzerland and one or two examples within the European Union. To conclude, the start and end points of copyright protection for a work are discussed. 344

675 These principles are widely accepted. They are statutorily anchored in Art. 9 and Art. 11 para. 2 Swiss CopA, Art. 4 para. 1 lit. b Computer Programme Directive, §11 ff. and § 69c para. 2 German UrhG, Art. 6bis and 12 RBC. In 17 U.S.Code § 106A the editing rights are statutorily only protected in the case of visual arts. However, as they are a signatory state of the Revised Berne Convention, they are obliged to offer this right to literary works as well. The Revised Berne Convention and the United States do not explicitly mention an author's right for first publication. It is however granted in Germany and in Art. 9 para. 2 Swiss CopA and § 12 para. 1 German UrhG.

676 See comprehensive discussion in: GRUETZMACHER, 552 ff.

677 See Art. 2 para. 5 RBC, Art. 2 para. 1 Swiss CopA, 17 U.S. Code § 102 lit a, § 2 para. 2 in conjunction with § 69a para. 3 German UrhG.

1. Subject Matter

345 In the first section, the classical protection subject of copyright is explained, focusing on creative and artistic works. A separate section then describes where the demarcations can be made for simple ideas and functionalities. This section also covers how these basic rules can be adapted to computer programs to determine which parts of software are covered under the copyright's subject matter.

a) Creative and Artistic Works

346 Copyright is intended to cover creative artistic and literary works.[678] Creativity or aesthetics in this context can refer to any work that is directly perceptible by the senses, that can be perceived with the eyes and ears and that evokes an experience or imagination in the consciousness.[679] What can be regarded as an artistic or literary work is explained in Art. 2 para. 1 RBC, including writing, lectures, dramatic or dramatico-musical works, choreographic works, musical compositions, cinematographic works, works of drawing, painting, architecture, sculptures, photographic works, works of applied art and maps. The extent to which the literary or artistic work in the individual case is eligible for copyright protection is not determinable by demarcating literature from art, but by examining whether the intellectual creation has an individual character.[680] Therefore, in order to be considered a creation the actual act of value creation, regardless of the form, is decisive.[681]

347 While Kummer in 1968, partly, and Troller in 1983, fully, denied the copyrightability of computer programs,[682] Cherpillod in 1985 and Rauber in 1988 closely evaluated their eligibility for copyright protection.[683] Today, software is explic-

678 Art. 2 para. 1 RBC, Art 1 Swiss CopA, 17 U.S. Code § 102 lit. a, § 1 f. German UrhG.

679 Troller (1967), 386 f. and 407 ff., with further references.

680 Barrelet/Egloff, Art. 2 N 7, with further references: Marbach/Ducrey/Wild, N 252

681 Rehbinder/Viganò, Art. 1 N 7; Nimmer/Nimmer (2016), N 2-6; Commentary on the German UrhG (Loewenheim), § 2 N 25 ff.

682 They considered computer programs as simple instructions to the human mind, and thus uncopyrightable (see Kummer, 200 ff.; Troller [1983], 356 ff.). Their classification must today clearly be regarded as superceded.

683 See Cherpillod (1985) and Rauber (1988); see also the remarks of Dreier in 1988, concluding that "reliable criteria" to identify copyrightable computer components did not exist at the time (Dreier (1988), 483).

itly covered as a copyrightable good on an international basis[684] having found its way into territorial copyright acts, including in Switzerland,[685] Germany[686] and the United States[687].[688] In Switzerland, the relevant regulations may be found in the Swiss Copyright Act. For the members of the European Union, there are a number of directives released by institutions of the European Union, including the E-Commerce Directive, the Copyright Directive and the Computer Program Directive.[689] All three directives do not have directly applicable sets of rules, but have to be implemented by the member states. On the other hand, in the United States, the U.S. Code only prescribes the most essential basics in the statute, mostly limited to what was requested of the United States as a member of the World Trade Organisation. Further guidance is offered in the Digital Millennium Act and, as a common law country, in jurisprudence.

348 The copyright protection of computer programs is two-fold: on the one hand, the international community has widely agreed to include computer programs as literary expressions.[690] It thereby protects constructivist thinking and the verbal capacity to express a creative program in a literary form,[691] which requires creative thought and energy[692]. In the 1980s' understanding, this early comprehension mainly included verbal instructions to the computer by individual human beings. In today's understanding, although not explicitly mentioned in the copyright acts, copyright protection is also partly open to artistic

684 The copyright quality of computer programs was recognized internationally at the beginning of the 1990s and they were integrated into the subject of copyright with Art. 10 para. 1 TRIPS Agreement and Art. 4 WCT.

685 Art. 2 para. 3 Swiss CopA.

686 § 2 para. 1 German UrhG.

687 17 U.S. Code § 102 lit. a refers to a machine or device.

688 Software was already being traded before the 1990s and was partially integrated into the territorial jurisdictions. In the earlier years however, no regulatory protection seemed necessary as fewer products were available and their trade was limited to public institutions such as researchers and academics, who relied on an informal exchange of helpful tools. See an interesting account of the historical development in HARISON, 68, with further references.

689 As they are incorporated into national law, they are not of such importance for this thesis.

690 See particularly Art. 4 WCT.

691 HOMMEL ET AL., 37.

692 PERELMAN, 922.

visual works, comprising illustrations, graphic works and visualizations.[693] Depending on the form of expression or component in question, copyright protection may include literary and visual elements under its subject matter.

b) Exclusion of Ideas and Functional Prerequisites

349 Copyright law also stipulates certain elements that are excluded from the subject matter of creative works:

aa) Ideas

350 Ideas are free to use and are therefore excluded from the subject matter.[694] Copyright under the so-called *idea-expression dichotomy* differentiates between ideas, which are not protected in copyright law, and their perceptible manifestation in an expression. An example of an idea in computer development is 'to create an online network, where people can create profiles and share content'. This basic idea can be implemented in different ways, it is called *non-expressive*. Today, many social media networks are based on this same idea but each service provider has realized it differently at a technical, visual and conceptual level. Only the perceptible implementation of an idea, in an *expression* with all its characteristics, is protected as a *work*.[695] Unlike in patent law and trade secrecy, idea-like shapes of a creation are widely excluded from its subject matter. This is regardless of whether the idea is entirely abstract or 'soft', or more detailed and specific, and thus 'hard'.[696] It is only if the author's idea is sufficiently materialized and becomes part of the final expression, that

693 Art. 2 para. 1 RBC.

694 Art. 9 para. 2 TRIPS Agreement.

695 See particularly Art. 9 para. TRIPS Agreement: "Copyright protection shall extend to expressions and not to ideas, procedures, methods of operation or mathematical concepts as such". The idea–expression dichtotomy originates from *Baker v. Selden*, 101 U.S. 99 (1979); see differentiation in *Whelan Associates v. Jaslow Dental Laboratory*, 797 F.2d 1222 (3d Cir. 1986); *Donaghue v. Allied Newspaper, Ltd.*, (1938) Ch 106; *Nutt v. National Institute, Inc.*, 31 F.2d 236 (2d Cir. 1929); *University of London Press v. University Tutorial Press*, 1916, 2 Ch. 601; *L.B. Plastics Ltd v. Swish Products Ltd.*, decision of the House of Lords of January 3, 1979, published in RPC, 1979, 551 ff.; *Designers Guild Ltd. v. Russell Williams (Textiles)*, decision of the House of Lords of November 23, 2001, published in ECDR, 2001, 10 ff.

696 SAMUELSON (2012), 162; SAMUELSON (2016), 430 ff.

aspects of this earlier stage are covered.[697] Know-how itself cannot be protected with copyright, but may be covered in an author's way of describing know-how in a work.[698] The subject matter of copyright is therefore limited to the particular, individual creative way of expressing an idea in a perceptible work. If a competitor expresses the same idea in a sufficiently different way, no infringement can be claimed.[699] According to the rationale behind this rule, granting strong rights on the basis of abstract ideas would reduce the public domain inappropriately compared with what society could profit.[700] As explained by the European Court of Justice, this protection would "monopolize ideas, to the detriment of technological progress and industrial development".[701] With the approach of only protecting expressions under the copyright scope, a balance between exclusionary rights and society's interests can be maintained.[702] The measure further intends to promote competition, access to information and freedom of expression.[703] As a consequence, copyright only covers the materialized idea in a concrete expression, which is why its scope of protection is quite small compared with that of patent law.[704]

This approach of copyright to distinguish between ideas and their expression presupposes that we can clearly separate one from the other. The distance between an idea and its expression, however, appears smaller when speaking of multimedia products, such as computer games, social networks and interactive visual interfaces.[705] A particular difficulty arises in the copyright assess- 351

697 Troller (1983), 351 and 374; Nimmer/Nimmer (2016), N 2-33 f. and 2-38 ff.; Dorr/Munch, 251 f. and 283 f.; Commentary to the German UrhG (Loewenheim), § 2 N 73; Samuelson (2017b), 1498 f.

698 *Baker v. Selden*, 101 U.S. 99 (1979), 104.

699 See: decision of the ECJ of May 2, 2012, C-406/10, SAS *Institute Inc. v World Programming Ltd.*, N 41.

700 This is especially under the premise that copyright protection today lasts a minimum of fifty years after a work's release; see also Fisher, 16.

701 Decision of the ECJ of May 2, 2012, C-406/10, SAS *Institute Inc. v World Programming Ltd.*, N 40.

702 Fisher, 16.

703 Samuelson (2016), 457 ff.

704 Calame (2006), 659; Hilti/Pedrazzini, 196 ff.; Staempfli Commentary on the PatG/EPC, Art. 1, N 57; Cherpillod (1985), N 122 f.; Nimmer/Nimmer (2016), N 2-31; Scotchmer (2006), 76, with further references; Cherpillod (2014), N 46 ff.; Wandtke/Bullinger (Gruetzmacher), § 69a, N 27 f.; Schwarz/Kruspig, 112 f.; Samuelson (2017b), 1498 f.

705 Straub (2002), N 2 and 3; see also discussion in Samuelson (2017b), particularly 1516 and 1536 f.

ment of algorithms implemented in codes.[706] The U.S. Court of Appeals recognized the difficulty of distinguishing between an idea and its possibly copyrightable expression and established the so-called *merger doctrine* which says that if the literary formulation or design of an expression is *dictated* by its idea, the idea and its expression are 'merged' and therefore inseparable.[707] For a merged expression, no copyright protection is available. The courts further explained in *Apple Computer, Inc. v. Formula International, Inc.*[708] that the purpose of a utilitarian work would represent the work's idea, and that everything that was not necessary to achieve and illustrate that purpose or function would be part of the expression of this idea. The decisive question in evaluating potentially copyrightable material is therefore *whether there are several possibilities to implement an idea.*[709] If there is only one or very few ways to realize an idea, the expression and the idea are compound, they are considered unprotectable.[710] As expressed in *Whelan v. Jawlow*,[711] if, however, "there are various means of achieving the desired purpose, then the particular means chosen are not necessary to the purpose; hence, there is expression and not idea".[712] We are thus looking for non-compulsory alternative methods to express a particular idea.[713]

352 *Nichols v. Universal Pictures*[714] then established that with the help of a so-called abstraction test, the examiner should verify by abstracting and filtering whether the working product resembled a general pattern, and thus a simple uncopyrightable implementation of an idea, or whether the implementation was sufficiently specified to represent a concept. In *Lotus v. Borland*[715] the court for the first time applied the abstraction test to computer programs and

706 See discussion in Brandi-Dohrn, 183

707 *Lexmark International, Inc. v. Static Control Components, Inc.*, 387 F.3d 522 (6th Cir. 2004); Apple Computer, Inc. v. Franklin Computer, Corp., 714 F.2d 1240 (3d Cir. 1983); Whelan Associates v. Jaslow Dental Laboratory, 797 F.2d 1222 (3d Cir. 1986); Herbert Rosenthal Jewelry Corp. v. Edward and Lucky Kalpakian, 446 F. 2d 738 (9th Cir. 1971); Morrissey v. Procter & Gamble Co., 379 F.2d 675 (1st Cir. 1967).

708 *Apple Computer, Inc. v. Formula International, Inc.*, 562 F. Supp. 775 (9th Cir. 1983).

709 *Lexmark International, Inc. v. Static Control Components, Inc.*, 387 F.3d 522 (6th Cir. 2004).

710 See discussion in Samuelson et al., 2358; Samuelson (2016), 426.

711 *Whelan Associates v. Jaslow Dental Laboratory*, 797 F.2d 1222 (3d Cir. 1986).

712 *Whelan Associates v. Jaslow Dental Laboratory*, 797 F.2d 1222 (3d Cir. 1986), c. 1.

713 See Boecker, 115, referring to *Lexmark International, Inc. v. Static Control Components, Inc.*, 387 F.3d 522 (6th Cir. 2004).

714 *Nichols v. Universal Pictures Corporation*, 45 F. 2d 119 (2d Cir. 1930); see similarly in Haberstumpf (1993), II N 60 f.

715 *Lotus v. Borland* 516 U.S. 233 (1996).

evaluated whether a particular hierarchy of a user interface's top menu structure should be considered as copyrightable. It assessed which form of listing represented a common pattern for an idea and where room for individual expression was available.[716] Similarly, in the United Kingdom the English and Welsh High Court in its evaluation considered whether there would be several different ways of producing a similar or identical result in a computer program. Using underlying ideas and principles, without copying the actual expression, e.g. its manifestation in the source code, would not represent a copyright infringement, as long as the second developer came up with his or her own design for the outcome.[717,718] At its core, both the merger doctrine and the approach of the English and Welsh High Court seem to be closely related to Kummer's *theory of statistical uniqueness*, which is used in Switzerland and in nearby German-speaking countries to determine the originality of works.[719] The bottom line in all three approaches is to look for a creative leeway to implement an idea. If such a scope of creation is available, the product represents a copyright-relevant implementation and is no longer just a bare idea.

bb) Functionalities

International copyright law focuses on the protection of *creative works*. Excluded from copyright protection are elements in a work that possess a direct associated function and are thus solely included in a work to fulfil this partic- 353

716 For practical considerations and more contextual information on the merger doctrine, the abstraction test and their application to computer programs, see later N 726 ff.

717 *Navitaire Inc. v Easyjet Airline Co. and BulletProof Technologies Inc.*, decision of the EWHC of July 30, 2004 (EWHC 1725).

718 The same would be true for computer programming languages, as Samuelson specifies. A programming language, although specific and detailed in its concept, may be implemented differently in the final source code. The programming language here only represents the idea and the semantic principles are then applied, conceptualized and realized in the final expression (Samuelson (2012), 162 f.).

719 The theory of statistical uniqueness (Theorie der statistischen Einmaligkeit) examines whether a work is individual enough so that it seems highly unlikely that the same or substantially the same work would be created by a third party for the same idea. If both answers are negative, a work is considered original (Kummer, 30 ff., 44 ff., 47 ff., 63 ff., 80, see particularly 67.; BGE 134 III 166, c. 2.5; BGE 130 III 168 – *Bob Marley*, c. 4.1 ff.; see also brief summary in Straub (2011), N 73 ff., with further references; von Bueren/Meer, N 179 ff.). For more details, see N 363.

ular purpose.[720] Functional aspects in a work are considered as necessary incidents, predetermined elements that people may need in order to achieve a certain result, and hence belong to the public domain.[721] Likewise, investing financial, labour or time resources does not alone suffice for copyright protection, as it represents a necessity related to the production of the work.[722] Although these elements do not harm the copyrightability of a work,[723] the creative element is the focus of copyright law and the work needs to exhibit an apparent creative influence or strength in order to be protected under this legal institution. Functional elements are particularly hard to distinguish from other types of utilitarian elements, which narrow the creative leeway of an author, but are not rooted in technical operational necessities, rather in another compelling reason to implement, such as socially predetermined demands of the target industry, e.g. the users.[724]

354 In the case of computer programs it is very difficult to determine beyond doubt whether an expression follows a merely creative or functional aspect. As Harison outlines in his work, with the incorporation of computer programs under copyright protection the subject matter of copyright has been extended to semi-functional and operational elements that are combined with creative aspects.[725] Computer programs show an immediate functionality, while also having an artistic quality.[726] They are not only perceived as classic creative work as they also pursue a concept of use, suggesting economic and technical features with an economic rationale. It is thus important to sort out the functional elements that steer the behaviour of a computer program or implement

720 See particularly the ruling of the ECJ of May 2, 2012, C-406/10, *SAS Institute, Inc. v. World Programming, Ltd.*; *Mazer v. Stein*, 347 U.S. 201 (1954); see also discussion in Nimmer/Nimmer (2016), N 2A-19 ff.; Dorr/Munch, 284 f.; Samuelson (2017a), 1267 ff.; Buehler, 91 ff.

721 *Baker v. Seldon*, 101 U.S. 99 (1979), 103.

722 In *Feist Publications v. Rural Telephone Service*, the court held that "the sweat of the brow" alone is not sufficient to bestow copyright, see *Feist Publications v. Rural Telephone Service*, 499 U.S. 340 (1991), c. 44 f.

723 Art. 9 para. 2 TRIPS Agreement explicitly excludes subjects that are only technical and do not involve creativity, such as technical procedures, processes, systems, methods and operations; see also decision of the ECJ of May 2, 2012, C-406/10, *SAS Institute, Inc. v. World Programming, Ltd.*, N 39; see also explanations in Samuelson et al., 2347 ff.; Cherpillod (1985), N 281 ff.; dissenting: Rauber (1988), 158.

724 For further information on otherwise predetermined elements that affect the originality in work creation, see N 364.

725 Harison, 176 f.

726 Straub (2013), 1.0 and 2.0; Reichman, 2477 f.

a utilitarian structure, and thus do not include creative leeway while being implemented into an expression. Instead, they are dictated by the specific context of the use of an element "to carry out a preassigned function".[727]

With the *scènes à faire doctrine*, the U.S. court law has established a helpful doctrine in the form of a thought experiment to figure out whether or not an element is functional. It also serves to identify other types of utilitarian or predetermined elements. The doctrine is based on the idea that every artistic work involves a number of elements or scenes in a presentation that are obligatory to illustrate a genre, such as indispensible stereotypes in certain movies.[728] As these elements are solely embedded to fulfil a task properly, and serve a solely technical purpose, they do not involve a creative step. In *Altai*,[729] the scènes à faire doctrine was adopted for computer programs. The court in particular held that certain mechanical specifications or compatibility requirements of a network would be dictated by external factors (the *scènes à faire*).[730] Since they are predetermined, they lack creative leeway, and hence cannot be copyrighted. What is to be considered as predetermined strongly depends on the individual setting and application case and therefore has to be evaluated for the particular situation. In computer programs, functional elements are often manifested through technical necessities, technical specifications or system requirements that are required to enable the program to function in its environment.[731] Similarly, the program flow, the command structure, certain expressed formulas or simply systemic constraints to continue with previously used elements in an established network may be technically predetermined in computer programs.[732] 355

c) *Current Interpretation of the Subject Matter of Software Copyright*

Copyright protection for computer programs thus involves the protection of literary and visual expressions to the extent that they do not serve a merely functional purpose or are dictated by an abstract idea. A case-by-case evalua- 356

727 *Oracle America, Inc., v. Google Inc.*, 872 F. Supp. 2d 974 (Fed. Cir. 2012), c. 200 f.

728 *Williams v. Crichton*, 84 F.3d 581, 583 (2d Cir. 1996), commenting on *Walker v. Time Life Films, Inc.*, 784 F.2d 44, 53 (2d Cir.); see also comment in SCHWABACH, 49.

729 *Computer Associates Int., Inc. v. Altai Inc.*, 982 F.2d 693 (2d Cir. 1992).

730 *Computer Associates Int., Inc. v. Altai Inc.*, 982 F.2d 693 (2d Cir. 1992), c. 2.b.

731 See also further information on the computer's system environment, in N 127 and N 733 ff. for practical considerations.

732 THOMANN (1998), 13.

tion is necessary to assess whether an element of a computer program can be regarded as copyrightable or not. With regard to the current interpretation in Europe and the United States, however, a brief general summary can be provided on the copyrightability of typical components in a software product:[733]

- The copyrightability of a compact software version, sold for download or on a CD, as opposed to *one-to-one* copies is undisputed.[734]
- Traditionally open for copyright protection are also the classic literary forms of expression of a computer program, such as the *source code*, the *object code* and the *associated documentation such as manuals and instructions*.[735] However, imitations of the literary expression have to be very similar to the original to be considered a copy. It is also unclear to what extent translating source code passages from one programming language to another are covered under copyright protection. While this problem is widely settled by law for classic literary works,[736] it is unclear to what degree the author's exclusive and absolute right to control their work's use is transferable to computer programs, and whether the translated source code in a new programming language represents an original

733 The subsequent listing is not exhaustive, but rather represents a list of the most discussed components under copyright. For an extended overview of the status quo in legal software protection, see also Straub (2011).

734 Decision of the Court of Justice of Geneva of August 6, 1986, published in SMI, 1987, 217 ff.; decision of the Court of the Canton of Zug of August 30, 1988, *Auto-CAD I*, published in SMI 1989, 58 ff.; decision of the OGer Zurich of October 11, 1990, published in SMI, 1992, 199 ff.; decision of the German BGH of September 20, 2012, I ZR 90/09, published in GRUR, 2015, 509 ff., and its discussion in Harison, 64 f. and 173 ff.; Reichmann, 2487; Marly, N 89; Koehler, 85.

735 Art. 10 para. 1 TRIPS Agreement; see for example: BGE 125 III 263 – Software-Lizenzvertrag; decision of the OGer of Zurich – *Software*, published in sic!, 2011, 230 ff.; decision of the ECJ of December 22, 2010, C-393/09; decision of the OLG Hamburg of February 29, 2012, 5 U 10/10, published in MMR, 2012, 832 ff.; decision of the OLG Frankfurt of January 27, 2015, 11 U 94/14, published in GRUR, 2015, 784; *Apple Computer, Inc. v. Franklin Computer, Corp.*, 714 F.2d 1240 (3d Cir. 1983); *Data Cash Systems, Inc. v. JS&A Group, Inc.*, 628 F.2d 1038 (7th Cir. 1980); Zirn, 33 f.; Wittmer, 114 ff. and 130 ff.; Calame (2007), 331; Thouvenin/Berger, 6/3.2, 1; Kummer, 203; Perelman, 928 ff.; Lehmann (1988), 2420; Thomann (1998), 12; Waltl, 47 ff., 59 ff. and 74 f.; U.S. Congress (1992), 13; Straub (2013), 2.2 and 2.3; Koehler, 87; Wandtke/Bullinger (Gruetzmacher), § 69a, N 10 f.; Nimmer/Nimmer (2016), N 2A-175 f.; Lemley et al., 35 f.; Commentary to the German UrhG (Loewenheim/Spindler), § 69a N 5; Samuelson (2012), 159.

736 See Art. 3 para. 2 Swiss CopA, § 3 German UrhG, 17 U.S. Code § 101.

work of its own. Apart from a regulation in the European Union, there is no prevailing opinion in either doctrine or case law in Switzerland or the United States in this respect.[737]

- While in the United States the conceptual *structure, sequence and organization of the source code* are protected by copyright, according to case law,[738] in Europe their integration is still unsettled. Although the literature and the Swiss Dispatch confirm their protection, the European case law has not explicitly confirmed this interpretation yet. The prevailing doctrine, however, generally assumes that they are eligible for copyright protection.[739] Similarly unclear is the fate of *programmed instructions, specifications and included parameters*. The definition of parameters and formulated specifications in a source code are probably covered,[740] but whether the simple linguistic formulation of a functional instruction is also copyrightable remains unclear.[741]
- Another question is how *computer assisting generators* (such as macros, listing, platforms, tools and libraries) and their products should be treated in copyright. While tools and generators are generally considered as creative works like every other original computer program, their output, i.e. source code created by assisting services, is currently assumed to be eligible for copyright only if human influence is recognizable in the final output. This means that the developer is able to affect the final output by making certain noticeable decisions in the creation of the tool, which

737 See for a more detailed discussion N 832 ff.

738 *Whelan Associates v. Jaslow Dental Laboratory*, 797 F.2d 1222 (3d Cir. 1986); *Lotus Development, Corp. v. Paperback Software International*, 740 F. Supp. 37 (District Court of Massachusetts (1990).

739 Dispatch to the Swiss CopA, BBl. 1989 III, 523. Due to the missing case law in this field, authors can only speculate on their eligibility. Supporting the copyrightability of the structure, organization and sequence: CHERPILLOD (1985), N 283; HARISON, 175 ff.; STRAUB (2011), N 81-83; STRAUB (2013), N 5; NEFF/ARN, 141 f.; WITTMER, 116 ff.; RAUBER (1988), 184 f.; WANDTKE/BULLINGER (Gruetzmacher), § 69a, N 23 ff.

740 STRAUB (2013), N 3 and 13.2.

741 See supporting examples: HEINEMANN (2005), 69, deduced from European Commission, decision of March 24, 2004 (case no. COMP/C-3/37.792) – *Microsoft*, c. 1003 f.; WITTMER, 117 f.; RAUBER (1988), 185; opposing examples: decision of the ECJ of May 2, 2012, C-406/10, *SAS Institute, Inc. v. World Programming, Ltd.*, N 39; CHERPILLOD (1985), N 283.

then shape how the tool produces the outcome.[742] Koehler and Kummer further argue that, mostly, the parts of the generator are then not used one-on-one but instead need to be adapted to the specific application environment or mixed with personal elements.[743] All these factors help with the status of the assisting structures.

- *Algorithms* are generally excluded from copyright protection, as they are regarded as ideas in the form of mathematical objects and therefore scientific information that belongs in the public domain.[744] This means that scientists and researchers are able to use algorithms for experiments, without copyright law affecting the competition for innovation.[745] In the domain of computer programs, algorithms are said to have a functional characteristic, representing a process in the form of a rule.[746] For both reasons, they can't be copyrighted.
- Mere technical *functions* and *features* fall under the public domain, being considered, as their name indicates, functionally utilitarian, used to fulfil a particular task.[747] Only their original implementation in a concrete expression, i.e. within a source code, may be copyrightable.[748]
- *Data* should be distinguished and considered separately from the computer programs that process them.[749] Data do become copyrightable as

742 See discussion in Straub (2011), N 67; Straub (2013), 2.1; Wittmer, 129 f.; Buehler, 55 ff.; Thomann (1998), 16; Commentary to the German UrhG (Loewenheim/Spindler), § 69a N 15; see no. 4.1.1 of the AIPPI Resolution in: AIPPI, 304; more critical: Koehler, 91 f.

743 See Koehler, 91; Kummer, 42 ff. and 199 f.

744 The Swiss Dispatch for the Copyright Act in 1989 declared algorithms as generally uncopyrightable (Dispatch to the Swiss CopA, BBl. 1989 III, 523). See more differentiated evaluation in: Wittmer, 67 ff.; Brandi-Dohrn, 180 and 183; Haberstumpf (1983), 234 f.; Rauber (1988), 165 f.; Calame (2007), 329; Wiebe, 64 ff.; Rehbinder/Viganò, Art. 2 N 17; Staempfli Commentary to the Swiss CopA (Cherpillod), Art. 2 N 64; Neff/Arn, 34 ff.; Wandtke/Bullinger (Gruetzmacher), § 69a, N 28; Commentary to the German UrhG (Loewenheim/Spindler), § 69a N 12; Widmer, 252; Samuelson et al., 2383 ff.; Newell, 1026 f.

745 Mathematical Programming Society, 3 f.; Newell, 1026 f.; Commentary to the German UrhG (Loewenheim), § 2 N 80 ff. and § 69a N 12; Rauber (1992), 42.

746 See descriptions in: Samuelson et al., 2383 ff.

747 Decision of ECJ of May 2, 2012, C-406/10, *SAS Institute, Inc. v. World Programming, Ltd.*; *Mazer v. Stein*, 347 U.S. 201 (1954); see also above N 353 ff. for functionalities in copyright.

748 See also Straub (2011), N 88, with further references; Dorr/Munch, 407 ff.; Samuelson (2012), 161.; Samuelson (2016), 438 ff.

749 Marly, N 25 ff.

part of software, if processed in the structure of a database[750] or when integrated into a copyrightable computer program.[751] *Databases* are only protectable if their composition or compilation exhibits originality. It is not given if their structure is a direct consequence of the content.[752]

– Visual elements such as *icons* and *graphics* are copyrightable.[753] The copyrightability of *user interfaces* is generally accepted, provided that they show the creative input of the creator in the design and their illustrative form does not only cohere to the functional aspects.[754,755] As a tool, the computer can also participate in shaping the process or outcome, as long as the human being is the creative decision-maker.[756] The user interface is copyrightable in its totality of all possible screenshots,[757] but also

750 Art. 10 para. 2 TRIPS Agreement; STRAUB (2013), N 1.2; THOMANN (1998), 19.

751 STRAUB (2013), N 1.2.

752 See for example *West Publishing Co. v. Mead Data Central, Inc.*, 616 F. Supp. 1571 (District Court of Minnesota 1985); STRAUB (2013), N 16.7 and 32 ff., particularly N 32.3.

753 U.S. CONGRESS (1992), 13; *Stern Electronics, Inc. v. Kaufman*, 669 F.2d 852 (2d Cir. 1982); VON BUEREN/MEER, N 296 ff.

754 *Apple Computer, Inc. v. Microsoft, Corp.*, 35 F.3d 1435 (9th Cir. 1994); *Broderbund v. Unison*, 648 F. Supp 1127, 1133 (Northern District of California 1986); decision of the OLG Karlsruhe of June 13, 1994, 6 U 52/94, published in CR, 1994, 607 ff.; see for example HILTY/KOEKLUE, 803 and 807 f.; KUMMER, 203; THOMANN (1998), 18; HEPP/MUELLER/HERMANN, N 1581 ff.; STRAUB (2013), N 5.1; NEFF/ARN, 147 ff., particularly 149, and 181 ff.; HARISON, 64; WANDTKE/BULLINGER (Gruetzmacher), § 69a, N 14, with further references; U.S. CONGRESS (1992), 17 f.; LEMLEY ET AL., 78 ff.; WITTMER, 108; SCHLATTER, III N 77 f. and 92 f.; opinions against the copyrightability of user interfaces: *Lotus v. Borland* 516 U.S. 233 (1996); *Digital Communication Association v. Softclone*, 659 F. Supp. 449 (Northern District of Georgia 1987); the European Court of Justice decided that user interfaces were not covered under the Computer Program Directive, but under the E-Commerce Directive (decision of the ECJ of December 22, 2010, C-393/09); Commentary to the German UrhG (Loewenheim/Spindler), § 69 N 7.

755 Functional in this context means that if the formulated instructions are the only reason why a user interface has a particular layout, and the user interface does not offer any further creative contribution, the functionality is predominant and the user interface is therefore an uncopyrightable work (see also SAMUELSON ET AL., 2352 f.).

756 Same argument in NEFF/ARN, 182 f.; SCHLATTER, III N 92 f.

757 STRAUB (2011), N 80; THOMANN (1998), 18; NEFF/ARN, 148; MARLY, N 92 f., with reference to the decision of the ECJ of December 22, 2010, C-292/09, published in GRUR, 2011, 220 ff. – *BSA/Kulturministerium*.

outstanding individual elements, such as excerpts or fragments, may be protected by copyright in a concrete form of expression, provided they are original.[758]

- Whether the copyright protection of visual elements also covers the *look-and-feel* is widely disputed. To date, no European plaintiff has successfully claimed the look-and-feel of his or her computer program. Only the combination of structural elements and the user interface standing alone have been protected.[759] However, U.S. case law suggests that, in theory, copyright protection of 'concepts and feels' is permissible, if a creative step is shown, i.e. if combined with a creative visual effect that exceeds mere functionality.[760] The copyrightability of a creative and innovative look-and-feel seems to be possible, but with a high threshold.[761]
- Protection of the *development documentation* for computer programs is also controversial. Some authors claim that they are covered by copyright protection.[762] The EU Computer Program Directive in its Art. 1 (outlined in consideration 7) also covers preparatory design work leading to the development of a computer program, suggesting that the development documentation may fall under this provision. But this interpretation falls

---

758 See discussion in STRAUB (2011), N 80, with further references; BARRELET/EGLOFF, Art. 3 N 5; NEFF/ARN, 112 f.; KUMMER, 77; Commentary to the German UrhG (Loewenheim), § 2 N 87.

759 See for example *Broderbund v. Unison*, 648 F. Supp 1127, 1133 (Northern District of California 1986), in which the combination of menu points and lists was granted copyright protection.

760 *Broderbund v. Unison*, 648 F. Supp 1127, 1133 (Northern District of California 1986); *Apple computer, Inc. v. Microsoft, Corp.*, 35 F.3d 1435 (9th Cir. 1994); *Lotus v. Borland* 516 U.S. 233 (1996).

761 See for a liberal interpretation: SAMUELSON ET AL., 2429 f.; PINHEIRO/LACROIX, 411 ff.; see also discussion of various U.S. case law and conclusion in STIGLER, 232 f.; NEFF/ARN, 153 ff.; for a critical view see: NIMMER/BERNACCHI/FRISCHLING, 631 ff.; WANDTKE/BULLINGER (Gruetzmacher), § 69a, N 14; Commentary to the German UrhG (Loewenheim/Spindler), § 69a N 7; BUEHLER, 94.

762 Supporting view: Decision of the OGer Zurich of June 30, 1983 – *Managertyp*, c. 2, published in SMI 1985, 224 ff.; Commentary to the German UrhG (Loewenheim/Spindler), § 69a N 5, with reference to considerations 1 and 7 of the Computer Program Directive; NEFF/ARN, 123 f.; WANDTKE/BULLINGER (Gruetzmacher), § 69a, N 8; STRAUB (2011), N 30, N 50 Fn 67 and N 62; STRAUB (2013), N 2.2 and 2.3; THOUVENIN/BERGER, 6/3.5, 2; HEPP/MUELLER/HERMANN, N 1558; KOEHLER, 64 ff.; WIDMER, 251; THOMANN (1998), 12 and 14; THOUVENIN/BERGER, Kap. 6/3.2., 1; CALAME (2007), 327; U.S. CONGRESS (1992), 13; see opposing view: WITTMER, 34 f.; VON BUEREN/MEER, N 291.

short. It is, firstly, unclear what may be considered as preparatory material. Samuelson here speaks of a program's structure, sequence and organization of the design.[763] However, those elements usually represent the earliest definitive work result of the development process, which is why their copyright protection tends to be accepted. Further design material should, however, be evaluated in terms of how materialized the concepts for later definitive products are and whether they contain abstract ideas and unprotected functionalities. It is also unclear to what extent abandoned ideas, manifested in sketches as working products of trial-and-error are covered. The current wording of the international copyright acts and their tight implementation in practice leaves a strong legal uncertainty as to whether an early work product such as development documentation is tangible and original enough to represent a creative work. Especially in the case of computer programs that commonly involve a lot of planning and conceptualization, a clearer anchoring in law would be desirable.

Overall, copyright offers protection for a number of components in a computer program. While the simple literary and visual expression in the source code, object code and the product documentation are widely accepted in Europe and the United States, the treatment of certain elements such as the user interface, look-and-feel, the conceptual structure and organization, as well as the development documentation are still unsettled. 357

2. Protection Requirements

If the work falls under the subject matter of copyright, one also has to assess whether the protection requirements are fulfilled. According to these, a work has to represent an intellectual creation and be original.[764] 358

a) Intellectual Creation

A work has to represent an intellectual creation. The term creation means that a work has to be developed or created, but not just discovered.[765] Further, the 359

763 SAMUELSON (2012), 159 f.

764 See Art. 2 para. 5 RBC, Art. 2 para. 1 Swiss CopA, 17 U.S. Code § 102 lit a, § 2 para. 2 in conjunction with § 69a para. 3 German UrhG.

765 Decision of the OGer Zurich of September 8, 2005 – *Girello*, published in sic!, 2006, 329 ff.; WOESTEHOFF, 17.

work has to have an intellectual character. This is the case if an idea originates from human will which is then expressed in the creation.[766] The human makes use of their free and creative choices in order to consciously decide how to create their work.[767] The required degree of intellectual activity should not be set too high,[768] as the criterion mainly serves to distinguish works from products of chance, simply discovered findings without contribution, and creations developed by machines or through artificial intelligence, without human influence.[769]

360 A computer or computer program is considered as a tool to create a literary or visual work.[770] The intellectual criterion requires that a human has had a *conscious controlling and decision-making influence* on the creation of the expression.[771] As software engineering usually involves a project manager who influences the concept of the expression and an engineer programming a set of instructions, the required level of human influence is usually reached without great practical difficulty. The criterion becomes more problematic when assessing products of artificial intelligence or software tools. But when talking about 'classic software products', the expectations on the level of conscious and deliberate control over the creative process and its specifications should not be set excessively high.[772]

766 Art. L111-1 French IP Coe ("L'auteur d'une oeuvre de l'esprit jouit sur cette oeuvre [...]"); BGE 70 II 57 – *Habla*, c. 2; Dispatch to the Swiss CopA, BBl 1989 III 521; BGE 130 III 168, c. 4.5; see also discussion in Troller (1967), 388; Commentary to the German UrhG (Loewenheim/Spindler), § 69a N 15 f.

767 BGE 116 II 351; decision of the ECJ, December 1, 2012, C-145/10 – *Eva-Maria Painer v. Standard VerlagsGmbH*; *Callaghan v. Myers*, 128 U.S. 617 (1888); *Burrow-Giles Lithographic Company v. Sarony*, 111 U.S. 53 (1884); *Lotus Development Corp. v. Borland International, Inc.*, 49 F.3d 807, 815 (1st Cir. 1995).

768 Barrelet/Egloff, Art. 2 N 6, with reference to BGE 59 II 406 – *Kunz*.

769 See particularly Straub (2001b), 3; von Bueren/Meer, N 170; Wandtke/Bullinger (Gruetzmacher), § 69a, N 32; Commentary to the German UrhG (Loewenheim), § 2 N 38 f. and § 69a N 15 f.; Widmer, 250, with further references.

770 *Alfred Bell & Co. v. Catalda Fine Arts, Inc.*, 191 F.2d 99 (2d Cir. 1951); Hilty (2010), N 84.

771 Commentary to the German UrhG (Loewenheim), § 2 N 40 f. and 45 as well as § 69a N 15 f.; Straub (2001b), 3; Thouvenin/Berger, 6/3.4, 2; Buehler, 54 ff.

772 See also Straub (2001b), 3; Buehler, 55 ff.

b) Originality (Individuality)

The intellectual creation also needs to exhibit what some statutes refer to as originality, others as individuality.[773] Both terms mean that the creation has to *differ from pre-existing works but also from what would be expected and considered ordinary for the particular category of work in question.*[774] A purely technical or routine performance, however skilled it may be, is not original from a copyright perspective.[775] We are thus talking about the required creative step or threshold a creation has to exhibit in order to be protected under copyright law.[776] The author has to offer *his or her own contribution*, in which their *personality or thought is expressed.*[777] This contribution may be based on the design but also on the planning, selection, screening, arrangement and structur- 361

[773] There are slight terminological differences: *Originality* refers to the circumstance that a particular intellectual creation contains an independent creative imprint, as the embodiment of an author's intellectual thought. It requires an individual intellectual idea as well as skill, judgement and labour (see BGE 113 II 190, c. I.2.a. and further explanations in HILTY (2010), N 91; TROLLER (1967), 390; KUMMER, 35 f.; DREIER (1988), 478 f.; *Feist Publications v. Rural Telephone Service*, 499 U.S. 340 (1991), decision of the ECJ, December 1, 2012, C-145/10 – *Eva-Maria Painer v. Standard VerlagsGmbH*, c. 166; WOESTEHOFF, 17 f.; DUTFIELD/SUTHERSANEN, 80). *Individuality*, on the other hand, refers to a work's uniqueness or peculiarity, meaning it has to differ from similar work and from what may be expected (BGE 130 III 168 – *Bob Marley*, c. 4.4.; Dispatch to the Swiss CopA, BBl 1989 III, 521; decision of the German BGH of November 14, 2002, I ZR 199/00, published in NJW, 2003, 665 ff.; KUMMER, 36; KOEHLER, 44 f.; NIMMER/NIMMER (2016), N 2-7; TROLLER (1983), 361; RAUBER (1988), 87 f.; WIDMER, 253 f.; VON BUEREN/MEER, N 178; CHERPILLOD (1985), N 95 ff. and N 219 ff.; REHBINDER/VIGANÒ, Art. 2 N 1 lit. c.; WANDTKE/BULLINGER (Gruetzmacher), § 69a, N 33). Because of the proximity of the definitions, the two terms in practice are widely used as synonyms (see discussion in BGE 130 III 168 – *Bob Marley*, c. 4.1.) or certain characteristics of both definitions are mixed or confounded (see discussion in HILTY (2010), N 91). According to presented opinion here, both terms contain elements relevant to the same protection requirement in copyright, which is why they are here used as synonymous. The following description also includes elements of both definitions, for this reason.

[774] Dispatch to the Swiss CopA, BBl 1989 III, 521; DUTFIELD/SUTHERSANEN, 79 f.; RAUBER (1988), 86 ff.; WALTL, 55 ff.; NIMMER/NIMMER (2016), N 2-7; THOMANN (1992), 30; Commentary to the German UrhG (Loewenheim), § 2 N 53; BUEHLER, 97 ff.

[775] Commentary to the German UrhG (Loewenheim), § 2 N 53, with reference to BGH, GRUR, 1003, 34 ff., 36, and others.

[776] Commentary to the German UrhG (Loewenheim), § 2 N 51 f.

[777] Decision of the ECJ, December 1, 2012, C-145/10 – *Eva-Maria Painer v. Standard VerlagsGmbH*, c. 166.

ing of the substance.[778] The greater the creative leeway to express an idea, the bigger the potential influence creativity can have on its elaboration. On the other hand, if the author has a limited creative margin to exploit, the level of originality required decreases and the requirement may be met earlier.[779] The overall expression of the design of a piece of work with all its elements remains the key factor.[780]

362 In general, the literal or visual expression of a computer program would likewise have to exhibit originality to be sheltered under copyright law. However, according to a ruling of the European Court of Justice, it would not suffice that the development of a computer program involved a diligent but routine type of labour and skill. Instead, additional originality had to be exhibited.[781] In similar terms, the German Federal Court of Justice held that "the skills of an average talented creator who strings together and assembles elements merely technically is not protected" by copyright.[782] In these decisions, therefore a *qualified threshold for the originality* of computer programs was required, compared to other types of copyrightable works.[783] Also in a Swiss decision of the Court of the Canton of Zug the court asked for particular originality characteristics the program had to exhibit.[784] This practice to require a particularly high threshold of originality for software works in copyright – as established by the German and Swiss courts – was criticized in particular by Straub, Neff, Rauber & Arn and Haberstumpf. According to the named authors, as well as in the opinion presented here, there is no apparent reason or justification why certain judicial bodies would expect a higher degree of originality from computer-related expressions compared with other categories of works.[785] In par-

778 BGE 113 II 306 ff., 309; decision of the OLG Frankfurt of November 6, 1984, 14 U 188/81 – *Baustatikprogramm*, published in GRUR, 1985, 1049 ff.; see a more detailed distinction in KUMMER, 42 ff.; THOUVENIN/BERGER, 6/3.4, 2 f.

779 BGE 130 III 168 – *Bob Marley*, c. 4.1; BGE 113 II 190 – *Le Corbusier*, c. 1.2.a; von BUEREN/MEER, N 182.

780 BGH, GRUR, 1981, 520 ff, 521.

781 Decision of the ECJ of December 1, 2012, C-604/10 – *Football Dataco Ltd etc. v. Yahoo! UK Ltd etc.*

782 Decision of the BGH of May 9, 1985, I ZR 52/83 – *Inkassoprogramm*, published in BGHZ 94, 276 ff.

783 See further discussions in DREIER (1993), 782 f.; RAUBER (1988), 191 ff.

784 Decision of the Court of the Canton of Zug of August 30, 1988, *Auto-CAD I*, published in SMI, 1989, 58 ff.

785 STRAUB (2001a), 813; NEFF/ARN, 130 ff.; RAUBER (1992), 36 f., DREIER (1988), 478; HABERSTUMPF (1993), II N 88; see also discussion in THOMANN (1992), 29 ff.; Commentary to the German UrhG (Loewenheim), § 2 N 59 and § 69a N 17 f.

ticular, there is no legal basis legitimating such a differentiation. For exeample, the European Union's Computer Program Directive does not distinguish between computer programs and other potential copyright works in Art. 1 para. 3, nor does it ask for an additional criterion or a particularly creative threshold for computer programs. As with any work creation, it should be significant how much room a creator possesses in making a creative decision, and how much independence they have in choosing the program's construction.[786] As the Swiss Federal Supreme Court emphasized for a different work category, originality has to be sought and determined for the individual work.[787] Or as Thouvenin puts it, the task specifies how much leeway an author has in the individual case;[788] the decisive factor therefore is to what extent the creator can then make creative decisions within the task to be solved and how he or she does it. The individual steps in the design of a computer program include numerous creative decisions, which are then perceptible in the final expression. For example, Koehler believes that simpler programming languages offer more variety and therefore more originality in coding, because the particular components are not so determined as when using more complex higher programming languages.[789] The originality may, however, also consist in an entirely new constructivist, literary or visual result.[790] Wittmer offers a whole list of possibilities to attain originality, including defining parameters, combining code sequences and introducing particular visual interfaces.[791] He particularly connects elements associated with the applied expertise and know-how of an engineer to measure originality; the greater an engineer's experience, the more forms of an expression can be elaborated and the more distinctive the formulation of a code.[792] It is important to note that in software engineering, an idea can often be implemented in different ways through programming, differing in the selection of key elements, the programming language used, the structure, program flow, and the logic of the program, etc.[793] There are also various ways that visual elements can be integrated and displayed in an external de-

786 Kummer, 30; Ensthaler/Moellenkamp, 158 ff.; Haberstumpf (1993), II N 39 and 88 f.; Wirth, 70; Zehnder, 66 f.; Staempfli Commentary to the Swiss CopA (Cherpillod), Art. 2 N 18; Troller (1983), 357; Calame (2007), 331 f.; Widmer, 254.

787 BGE 130 III 168 – *Bob Marley*.

788 Thouvenin (2008a), 64 f. and 69.

789 Koehler, 87.

790 Harison, 185.

791 Wittmer, 114 ff.

792 See for example Wittmer, 120 f.

793 See for example Thomann (1998), 9.

sign.[794] The exact scope of originality that an expression may exhibit hence depends on the components in question. The High Court of the Canton of Zurich decided that a sequence of commands contains originality if it has not been taken from existing programs and if it does not solely represent the result of a routine performance.[795]

363 In practice, a work's originality or individuality is usually *measured against pre-existing works and what is known or established for a specific sector of art*. While several different testing approaches exist, for copyright protection in Switzerland but also in neighbouring countries the *theory of statistical uniqueness* (Theorie der statistischen Einmaligkeit) prevails. Introduced by Kummer[796] and supported by the Swiss Federal Supreme Court[797], the approach examines whether, in comparison, a work is individual enough so that it seems highly unlikely that the same or substantially the same work would be created by a third party for the same task.[798] If this is the case, copyright protection should be granted. The decisive factor is not the purely statistical or quantitative unique existence of an event or thing, but the statistical uniqueness in the work's creation, which must stand out from the usual. A creation is not unique if it is highly probable that the same or essentially the same creation would result from the same task.[799,800] The distinction between two works must also be sufficiently perceptible, which is not the case if the creation corresponds in all

794 Moehring, 273 f.; U.S. Congress (1992), 17 f.; Ulmer, 17 f.

795 Decision of the High Court of the Canton of Zurich of October 11, 1990, published in SMI, 1992, 199, quote from 202.

796 Kummer, 30 ff., 44 ff., 47 ff., 63 ff., 80, see particularly 67.

797 First of many: BGE 130 III 168 – *Bob Marley*, c. 4.4.

798 BGE 134 III 166, c. 2.5; decision of the OLG Hamburg of March 12, 1998, Az 3 U 226/97 – *Computerspielergänzung*, published in CR, 1998, 332 ff.; decision of the OLG Munich of May 27, 1999, 6 U 5497/98, published in CR, 1999, 688 ff.; see also a brief summary in Straub (2011), N 73 ff., with reference to Kummer, 30 ff., 47 ff., and 80; Wandtke/Bullinger (Gruetzmacher), § 69a, N 34.

799 BGE 134 III 166, c. 2.3.1; BGE 130 III 168 – *Bob Marley*, c. 4.1-4.4; see also Thouvenin (2008a), 69 ff. regarding the quantitative and qualitative individuality a work has to show in comparison with other works.

800 As Straub explains, the doctrine of statistical uniqueness is particularly useful for obvious and clear cases of either literary infringements, where the sequence and number of digits and letters can be analysed, or for two-dimensional visual expressions that constitute a particular combination of pixels, which can be statistically evaluated on their likeness. However, in other cases where we have to qualitatively evaluate the originality of two works as well as the difference between them, there is a lack of effective, justiciable criteria for making a reliable assessment (see Straub (2001b), 5).

parts to the usual.[801] Under the German interpretation, the work's individuality is expressed by its independence, uniqueness and individual character in its imprint by the author.[802] According to the German Federal High Court of Justice, originality can only be affirmed where a *work stands out from the mass of the everyday, the usually put forward, the banal, in short from what everyone would have produced in a similar way*.[803] The doctrine should therefore not be applied mathematically, but requires evaluative prudence with the specific case to be assessed.[804] Usually routine craftmanship that could be anticipated is not statistically unique. Comparably, in the U.S. with the *sine-qua-non-originality approach*, the original elements in a work are set in proportion to the pre-existing material, whose protection scope it shall not affect in any way.[805] If, on removing from the to-be-examined work what was known from previous works, including their contribution, no further original aspects are shown, the work is not copyrightable.[806] *The work thus needs to differ sufficiently from previous creations in order to be distinguishable*.[807] It has to represent a *substantial variation* from the underlying work.[808] In practice, all three approaches are similar in result; the decisive factor under all three theories is whether the author had creative leeway to vary.[809]

---

801 BGE 134 III 166, c. 2.3.2.

802 Troller (1983), 361 f.

803 Decision of the BGH of May 9, 1985, I ZR 52/83 – *Inkassoprogramm*, published in BGHZ 94, 276 ff.; decision of the BGH of April 17, 1986, I ZR 213/83 – *Anwaltsschriftsatz*, published in GRUR, 1986, 739 ff. On the basis of a decision of the German BGH, Rauber offers an alternative, three-tiered evaluation to assess the originality of works in computer programs (Rauber [1988], 191 ff., referring to Decision of the BGH of May 9, 1985, I ZR 52/83 – *Inkassoprogramm*, published in BGHZ 94, 276 ff.): First, those elements of a program which should be tested for originality are defined; second, the distinctiveness of the available program is compared with pre-existing works; in the final stage, the findings are tested to see whether they are 'expected' or 'routine', or whether they are unconventional and thus original (for this, the results of step 1 are used). Rauber's test shows that an individual valuation and judgement of what may be considered as 'routine', 'ordinary' or 'conventional' has to be made for each application.

804 For the latter statement, see also Troller (1983), 363.

805 *U.S. Auto Parts Network, Inc. v. Parts Geek*, LLC, 692 F.3d 1009, 1015 (9th Cir. 2012).

806 *ABS Entertainment, Inc. v. CBS Corporation, et al.*, case no. 16-55917 (9th Cir. 2018).

807 *Massachusetts Museum of Contemporary Art Found., Inc., v. Buchel*, 593 F.3d 38, 65 (1st Cir. 2010); *ABS Entertainment, Inc. v. CBS Corporation, et al.*, case no. 16-55917 (9th Cir. 2018); see also comment in Nimmer/Nimmer (2016), N 3-7 f.

808 *Woods v. Bourne Co.*, 60 F.3d 978, 990 (2d Cir. 1995).

809 Staempfli Commentary to the Swiss CopA (Cherpillod), Art. 2 N 27 ff.

364 A software product's originality may further be *restrained by the actual problem it wants to solve and the audience it wants to serve*. Similar to the framework of technical specifications or system requirements,[810] the destined social context in which the software is to be applied may limit the decision-making scope of the developer so that the author's room for creativity is attenuated.[811] This may especially be the case for particularly common or widely established software usability design principles. Such standardized utilitarian elements are indispensable for the software developer,[812] and in effect limit the author's creative margin, leaving less room for manoeuvre.[813] In *Altai*,[814] the court likewise held that a target industry's demands or widely accepted programming practices within the software industry might also represent external factors that constrain an author's creative leeway.[815] Where the developer's scope for action is narrowed by predetermined or procedural inevitabilities, the originality of expression is likely to be reduced.[816] In practice, the *scènes à faire* doctrine[817] may also in this case help to identify utilitarian elements and establish what elements in a work are obligatory for illustrative purposes to achieve a certain setting, to properly realize a project or fulfil a particular task.

365 The originality of a work also has an effect on the evaluation of *derivative or dependent and similar works*.[818] Derivative works in general are expressions that include substantial elements of a pre-existing work. The question is to what degree derivatives or similar works represent an independent work of

---

810 See also above regarding copyright-excluded functionalities, N 353 f.

811 See decision of the Court of the Canton of St. Gallen of May 24, 2005, DZ.2002.3, c. III.4.c; see also CHERPILLOD (1985), N 5 ff. and N 213 ff.; THOMANN (1998), 13; STRAUB (2001b), 5; STRAUB (2013), N 5.2; CARLETON, 408, 427 ff. and 431; ULMER, 18 f.; WIRTH, 70 f.; ZEHNDER, 66 f.; REHBINDER/VIGANÒ, Art. 2 N 1 lit. c.; HABERSTUMPF (1993), II N 90; SCHWABACH, 49; KOEHLER, 66 ff and 77 ff.; KUMMER, 47 ff.; Commentary to the German UrhG (Loewenheim), § 2 N 56; SAMUELSON (2017a), 1218 f. and 1267; REICHMAN, 1134.

812 *Williams v. Crichton*, 84 F.3d 581, 583 (2d Cir. 1996), commenting on *Walker v. Time Life Films, Inc.*, 784 F.2d 44, 53 (2d Cir.); see also comment in SCHWABACH, 49.

813 Consequently, if an unusual and creative way of implementation is chosen within a small creative scope to solve a particular task, the expected degree of originality is achieved with few design decisions, while a longer chain is required for obvious decision results (STRAUB (2001b), 6; THOUVENIN (2008a), 65 ff.; Commentary to the German UrhG (Loewenheim), § 2 N 57 f.).

814 *Computer Associates Int., Inc. v. Altai Inc.*, 982 F.2d 693 (2d Cir. 1992).

815 *Computer Associates Int., Inc. v. Altai Inc.*, 982 F.2d 693 (2d Cir. 1992), c. 2.b.

816 For practical integration and distinction of this topic, see later N 733 ff.

817 See above, N 355.

818 See similar reasoning in: NIMMER/NIMMER (2016), N 2-8; DORR/MUNCH, 273 f.

their own and when do they represent an infringing copy of an original. In general, an author has the right to protect his or her work from being distorted and changed, if a third party tries to do so without their consent.[819] This includes any change in the program as well as a controlling power to decide on upgrades, maintenance and adaptions to an author's computer program.[820] There are limits to the author's right to control changes to his or her product: Reuse of copyrighted products is allowed if a *visible distance* to the original is achieved; if the original is still *recognizable* in the derivative work, the author of the derivative work needs the consent of the original work's right holder for any kind of editing or imitation.[821] Similar or recognizable in this context means characteristics and distinctive creative elements that are preserved in the second work.[822] This is also applicable to parts, blocks and fragments of an original.[823] If, however, the original served only as a source of inspiration for the derivative work, and the original got further developed in terms of functionality, content and design until it figuratively faded and was no longer recognizable in the derivative, the author of the derivative work is free to use it. The central issue here is therefore the question to what extent the dependent author offered their personal contribution in the derivative.[824] The greater the strength of originality of the first work, the fewer original elements of it may be adopted for a later work.[825] In Switzerland and Germany, the practice of *libre utilisation* (Freiheit der Benutzung; free usage) has been adopted. Hereafter, the free use of a copyrighted work or work element is permitted if the adopted elements appear to be of minor importance with regard to the indi-

819 Art. 11 Swiss CopA, § 3 German UrhG, Art. 6bis and 12 RBC, for visual arts see also 17 U.S. Code § 106A. As the United States is a signatory of the Revised Berne Convention, they are obliged to offer this right to literary works as well.

820 Boecker, 151.

821 See art 3 para. 1 Swiss CopA, § 3 German, § 23 and § 24 para. 1 German UrhG; *Nichols v. Universal Pictures Corporation*, 45 F. 2d 119 (2d Cir. 1930), 121; see also Hilty (2010), N 122 ff.; Perelman, 944; Krueger, 131.

822 Decision of the LG Cologne of September 24, 2008, 28 O 530/05, N 75; decision of the Court of the Canton Vaud of April 10, 1987, published in SMI, 1990, 81 ff.

823 Decision of the Court of the Canton Vaud of April 10, 1987, published in SMI, 1990, 81 ff.; see also comment in Rauber (1992), 38 and 44 f.

824 See decision of the OGer Aargau of July 31, 1990, published in: SMI, 1991, pp. 79 ff.; see also comment in Rauber (1992), 44; Thomann (1998), 15 f.

825 Straub (2001b), 7.

viduality the new work shows, i.e. if its originality is comparatively weak.[826] If the second-hand work, however, represents an intellectual creation and shows enough originality of its own, it is no longer considered as a derivative but rather as an independent work, protected with its own intellectual property right.[827] In the case of computer programs, the Court of the Canton of Aargau in Switzerland ruled that if a computer program differs from an original by nine-tenths of the source code, a case of free usage would result.[828] In comparison, the U.S. interpretation evaluates whether two works show a *substantial similarity* if paralleled.[829] This means that a substantial, vital and essential part is equivalent in both works.[830] Both principles – the visibility criterion in Germany and Switzerland and the substantial similarity theory in the United States – in practice have a largely equivalent result. The more originality a derivative work can exhibit, the smaller the substantial similarity and the visible likeness between the two works. The evaluation as to whether a piece of work exhibits enough originality relies on discretion.

366 To conclude, an expression in a computer program has to exhibit originality in order to be covered by copyright. It has to be distinctive enough from what is known or could be expected. The creator thus has to make use of the available creative leeway in forming his or her piece of work. Predetermined characteristics in an expression constrain the amount or quality of creativity an author can use in his or her work. Originality is not only important to justify a work's eligibility for copyright, but also to set the copyright's boundaries in evaluating derivatives. It is thus necessary to evaluate on a case-by-case basis how much originality is contained in an expression and whether this suffices for independent copyright protection.

826 *Hawkes & Son, Ltd. v. Paramount Film Service, Ltd.*, decision of the EWCA, I CH 593 (1934); *Computer Associates Int., Inc. v. Altai Inc.*, 982 F.2d 693 (2d Cir. 1992); see also § 24 para. 1 German UrhG; see also discussion in Straub (2013), N 30.3, with reference to BGE 125 III 328 c. 4c and others; Cherpillod (1985), N 237 ff.; Barrelet/Egloff, Art. 3 N 5 and Art. 11 N 10; Neff/Arn, 217 f.; Landes/Posner (2003), 42 f.; Commentary to the German UrhG (Loewenheim), § 24 N 10;

827 See Art. 3 para. 3 Swiss CopA, § 24 German UrhG; Hilty (2010), N 123; see also discussion in Landes/Posner (2003), 42 f.

828 Decision of the Court of the Canton of Aargau, published in SMI, 1991, 85 ff.; see also discussion in Barrelet/Egloff, Art. 3 N 12.

829 *Computer Associates Int., Inc. v. Altai Inc.*, 982 F.2d 693 (2d Cir. 1992); see particularly Nimmer/Bernacchi/Frischling, 625 ff.; Nimmer/Nimmer (2016), N 3-3 f. and N 3-11 f.; in my opinion, the same may be assumed for Switzerland and the European Union.

830 *Hawkes & Son, Ltd. v. Paramount Film Service, Ltd.*, decision of the EWCA, I CH 593 (1934).

3. Term of Protection

The term of protection refers to the period during which the legal effects of a copyright last. The term starts at the moment when the work may be regarded as a copyrightable property and continues until it has reached its expiration date. During this period, the copyright holder can profit from his or her exclusionary rights and commercialize their work. Once the expiration date is reached, the property can theoretically still be commercialized. However, other market players are free to use, apply and commercialize the work without the author's consent. 367

A peculiarity of copyright from the perspective of intellectual property rights is that copyright protection is obtained *automatically* with the creation of a work. A further constitutive formality, as with a registry entry in patent law, is neither requested nor allowed for copyright.[831] The advantage of copyright therefore is that it does not involve any additional legal or administrative costs to register such as with other intellectual property rights. Some jurisdictions, including the United States, however, ask for a formal registry entry to undertake a copyrighted work in court processes.[832] The recognition of the work itself remains automatic. As no formal application is necessary, a work's copyrightability can only be tested substantively in the case of a legal dispute, where a court has to assess the copyright requirements.[833] At the same time, a court dispute usually is conducted *inter partes*, which is why a ruling's effect is limited to the parties involved in litigation.[834] 368

The copyright holder can execute his or her rights to the work exclusively until the end of the protection term is reached. For most work categories, the term of protection is internationally set at a minimum of fifty years after the last author involved in a work's creation has died.[835] Many legislations, such as Switzerland, the United States and several member states of the European Union, go beyond the minimum prerequisites of the Revised Berne Convention and generally grant copyright protection up to seventy years after the author 369

831 See Art. 5 para. 2 RBC.

832 17 U.S. Code § 411 lit. a.

833 See HILTY (2010), N 81; see also discussion in: THOMANN (1998), 12.

834 Its effect is *ex nunc* and *inter partes*.

835 Art. 7 para. 1 RBC. Tying the term of protection to the author and his or her death symbolizes the maintained connection between the author and his/her work which protects the author's heritage.

has died.[836] However, as a significant and symbolic exception to this, a shorter term of protection of only fifty years after the author's death is granted for computer programs under Swiss copyright law,[837] hence distinguishing computer programs from other work categories. Only very few statutories provide that the expiration is linked to the release of a specific work rather than the author's death,[838] and neither Swiss, German nor U.S. copyright law fall into this category.

VI. Conclusion

370 This chapter has provided an introduction to the most important principles in computer program protection. Software protection today involves various legal institutions, which each cover different aspects of software engineering and management. Most relevant for this study are copyright and patent law, whose protection criteria were discussed in greater detail in this chapter. It was shown that for these IP rights, many obligatory rules are provided on an international level and that the three regions – Switzerland, the United States and the European Union – offer a legally unified and, in practice, very similar position on many legal issues. Differences in law and practice were highlighted for each institution. Patent law in general covers technical innovations, and while computer programs are by word of the law excluded from protection within the European Patent Organisation, they are practically eligible for patent protection in all discussed jurisdictions to the extent that the software innovation offers a novel technical teaching whose industrial application is sufficiently disclosed in a patent claim. In the past, patent law protection, if available at all, focused mainly on algorithms and business methods. Copyright, on the other hand, protects original, non-functional artistic and creative expressions in their visual or literary form. It represents the main legal institution for international software protection. In the past, copyright protection was judicially affirmed for one-to-one copies of software products and for certain software components such as the source and object code, including

836 17 U.S. Code § 302 lit a, § 64 German UrhG and Art. 29 para. 2 lit. b Swiss CopA. See also the EC Copyright Term of Protection Directive which states in Art. 1 para. 1 that all countries of the European Union, i.e. Germany, have to guarantee a minimum term of protection of 70 years after the author's death. Due to consideration 15 of the Directive, this is also applicable to computer programs.

837 Art. 29 para. 2 lit. a Swiss CopA.

838 See in particular Art. 7 para. 4 RBC for photographic works and works of applied arts, setting the minimum term to at least twenty-five years from the making of such a work.

their structural and organizational elements, and visual exhibits such as the graphical user interface. The exact scope and application of patent law and copyright in the field of software is still not completely settled.

Chapter 5: Findings of the Interview Series

371 For my research I conducted 12 interviews in five months. The general feedback to my research topic and scope was very positive – almost all companies contacted were willing to give an interview. Depending on how detailed the answers to the questions were, a single interview lasted between 75 and 150 minutes.

372 The result is a list of claims and statements. In the following, the findings from the interview series are arranged by topic. In order to facilitate the reading, the sociological notes and interview material were greatly condensed. Where a clear tendency could be observed, a general statement was formulated. In this case, no individual source was cited because the statement applied to the majority of the interviewed companies and thus serves as a general assumption for the present study. Where particularities or contradicting claims occurred which appeared relevant or notable, statements were indicated separately, marking their source.

373 The first four sections of the findings (I-IV) discuss the technical and practical aspects of software development and commercialization. The fifth section (V) focuses on the legal scope of software protection, including the interviewees' experiences with the current system and their expectations of regulatory changes. In the final section (VI), we discuss the candidates' experiences with enforcing intellectual property rights in the market.

I. Software Development

374 The first section focuses on the development of computer programs. We here evaluate the meaning software engineering has for the industry, how the involved parties approach a software project and how they structure the development project.

A. The Relevance of Software Engineering

375 Software engineering was shown to be of high relevance for all the interviewed companies. Many interviewees emphasized the importance of software engineering for their businesses by describing it as their "core business", a "vital part of their work" or explaining that "without software their company would not exist". This observation was true for every company except one, which

stated that software engineering had only a minor role in their business.[839] All the other companies agreed that software engineering was one of the most important aspects of their business activity.

Several interviewees observed that during the last two decades the relevance of software has generally increased as their usability for industrial manufacturing, infrastructure and services has been augmented. They emphasized that mechanical governors and electronic circuits, which had previously been used in the industry primarily to control and steer machines or hardware products, were being replaced by software systems and computer programs.[840] This increase in relevance could also be observed in the employee structure of the companies; one interviewee noted that a quarter of all employees of his large company were working in the IT department doing software engineering.[841] He said that without computer systems, their service would not be operable.[842] Others claimed that every technical or machine process in their work was controlled and organized with software.[843] Although the statements were not verifiable, it suggests that companies tend to put a large share of their resources and budgets into software engineering, be it to develop software themselves or to edit and monitor bought third-party products. Computer programs today are used similarly to mechanical components in earlier decades to realize technical solutions and to fulfil and control particular processes.[844] Computer programs therefore have gained a significant importance for companies and institutions.[845] 376

B. The Development Process

In order to develop an adequate intellectual property system for computer programs, I believe it is important to understand what the course or development process of software looks like, which factors influence it and how long the process requires to be completed. 377

839 Transcript E, N 7. This feedback may be questioned as this company is one of the major patent applicants for software methods in the world. Possibly the interviewee was focusing on what part classic engineering plays in his daily work, he may have misunderstood the question, or it was his personal opinion.

840 See transcript E, N 3 and transcript G, N 11.

841 Transcript H, N 7.

842 Transcript H, N 7.

843 See transcript F, N 7, transcript G, N 11, and transcript A, N 10.

844 See, for example, transcript I, N 18 and 62, and transcript G, N 11.

845 See transcript G, N 11.

1. The Process

378 As described by the interviewees, a software development project does not differ greatly from other types of projects.[846] Usually, common project methods are integrated and adapted to the needs of the individual project structure. The interviewees distinguished several different methods to shape a software project.[847] They described how the selection of a method depended on available human resources and on the technical knowledge and domain know-how that is available in a company.[848]

379 The interviewees described several distinct phases or steps in each development project. They commonly mentioned a phase "where the idea pops up"[849] and you start working on it. This is what I have previously referred to as ideation. In a second step, the idea would be advanced and thought through carefully by setting up small test structures from drafts – the conceptualization phase. In this phase, the essential requirements and features are developed and elaborated in order to draft a first rough structure of the program.[850] It is during this phase that most of the research and preparation takes place and potential user needs are worked out. A considerable amount of resources are put into conceptualization because most of the coordination, documentation and business analysis is required at this stage in order to further the project.[851] When the design framework is finally established, the requirements and

846 Transcript G, N 25. Newer trends show that forms of project organization which were first developed for and tested in software development have now been adopted in other fields, e.g. holacracy and agile frameworks such as Scrum.

847 Transcript I, N 43.

848 Transcript I, N 40.

849 The ideation phase also refers to project ideas which do not merely pop up but instead emerge after market analysis or as a consequence of a business or client need. The common starting point for a software project is an idea or a formed business need from the clients. Most companies said that they develop in close cooperation with existing clients or the consumer community (see for example transcript C, N 15, transcript B, N 18, and transcript D, N 20).

850 Some interviewees split the design phase into two separate phases where first a rough concept is drafted with the requirement formulation and analysis and in a second phase a more detailed design is shaped where the structure is outlined.

851 The interviewee explained: "What we produce is software. But (...) comparing the workshare needed for coordination, documentation, business analysis and all the rather non-technical issues to the workshare for engineering, more time is requested for non-technical issues" (transcript I, N 14).

features are then realized technically in the engineering phase, where they are transferred into a computer-readable code form. After that, the software is implemented, tested and maintained in its future environment.

During the interview series it became clear that all of the companies tend to work with individually structured development models. They either follow the classic linear approach[852] or a version of the spiral[853] approach, depending on the complexity and particularities of the software project. Although both models are used regularly in all the interviewed companies, agile approaches in particular are said to be slowly replacing the classic linear waterfall approach; a project manager and his or her team of engineers work in shorter intervals or cycles, so-called *sprints*, in which one or more software parts or modules (e.g. a particular functionality) are developed and realized.[854] The software developers described how it was much easier to test a single increment or component instead of waiting until the whole product was completed. As the developer is able to create a first testable component sooner, the approach is very resource-saving and the project owner (and his/her team) can react faster to contingencies.[855] This first incremental output may function as a deliverable prototype that can be shown to potential clients, investors and other relevant people, which is interesting from a marketing and commercialization standpoint. The interviewees referred to this first deliverable as the *minimum viable product*: the earliest point in development when a product is apparent and able to stand alone.[856] From then on, new functions or adaptations – *deliverables or increments* – can be added piece by piece. At the same time, however this approach carries the risk that during a set of sprints a chain of unaudited or mismatching puzzle pieces may be created. The product owner thus 380

852 If a project is developed linearly the engineer works similar to a waterfall from one step to another without ever returning to the previous one. For more information regarding the linear approach, see N 172 ff.

853 Instead of working from one phase to another and having a final product at the end of a cycle, engineers with this approach focus on one specific function they want to develop first. They elaborate the requirements for this particular function and code until they have a small computer program (or several) expressing this function. It is like one piece of the final software. The engineer then jumps back to the stage where they select a new function they want to implement and then elaborate the requirements for this function and code it into another puzzle piece. For more information regarding the iterative approach, see above N 175 ff.

854 See explanation in transcript K, N 22.

855 See for example transcript D, N 83, where the interviewee describes the linear process as something you "used to do".

856 It is also referred to as the minimum viable increment or the minimal marketable product.

has to carefully coordinate the single increments created by several teams in parallel sprints, which may represent a disadvantage compared with the linear approach.

381 Although working with iterative models has become very popular, most companies do not dictate which approach their software developers should use. On this question, the engineering team is mostly free to decide for themselves. As a consequence, what we see in practice is that one software system that is designed by several engineers or teams may contain multiple distinguishable computer programs, where some were developed linearly while others were built iteratively.[857] An even stronger tendency can be observed for a mix or combination of both models where certain aspects of the linear model and the iterative model are joined.[858] In the case of such *hybrids*, engineers tend to follow a linear process from the idea to the concept where the main requirements and features are summoned, before they start programming iteratively and implementing one drafted feature or function after another. This finding therefore contradicts the previous understanding.

382 Another tendency observed is that companies are increasingly trying to think of software as a product that is developing continuously. During the earlier periods of software commercialization, computer programs were distributed on floppy disks and later on compact disks (CDs). In order to get a new version of the software, a new release had to be purchased on a new separate physical carrier. Today, most programs run on an online server and can be updated from the Internet. Maintenance of software products therefore is no longer concentrated from major version to major version but instead occurs continuously. Provided that a release runs as desired, the online users are not even notified about the ongoing process or changes, but instead should be able to use the software product without major interruptions to the system. This change in the development and implementation of software has a remarkable effect; new functions and features, updates and bug-fixing represent an ongoing process and are no longer clearly separable and identifiable product versions.[859] This newer release trend is called *continuous delivery*.[860]

383 In 2015, when the interview series was completed, only a handful of papers were found that described the continuous delivery approach from a computer

857 See for example transcript I, N 36.

858 See for example transcript H, N 17, and transcript A, N 28.

859 See transcript D, N 20, and transcript A, N 14.

860 For more information regarding the continuous delivery method, see above N 182 ff.

scientific perspective. Today, the online search engine of the Association for Computing Machinery (ACM) provides around 30,000 search results in its library for the term, most papers published between 2016 and 2018.[861] In 2019, as back in 2015, the continuous delivery method in software engineering practice seems to be growing in popularity and is widely used to publish software products on servers, offering groundbreaking possibilities for the ICT industry. The major benefit of this approach is that the users do not need to stop using a computer program at some point and purchase a new version.[862] As the update can be provided automatically, the user can install the new version without great difficulty. As no separate purchase is necessary and the software can be installed without a contribution of the users, the developer can change, adapt and release their products whenever they are ready to do so. The engineers can react faster to regulatory changes, current trends or new needs and provide the new version at shorter intervals instead of releasing big chunks of software on a new physical carrier every couple of years. The pace of software launching has thus picked up speed.[863] One simple component is constantly altered and renewed with short release breaks.[864] On the negative side, the engineers have to perpetuate backward compatibility and thus have to continue with the previous release's software system in order to integrate new modules or offer updates.[865]

While smaller companies tend to decide for each new software project which approach they want to follow for the development (*bottom up*), bigger companies rely *on standardized project processes* or road maps, which guide them through their development process (*top down*). These include clearly defined, separable phases and steps. Larger companies also incorporate legal counsels in their processes, which are at the disposal of the project developers and engineers.[866] The legal counsels retain an overview of the whole development process from its beginning to its end. More extensive legal monitoring is inte- 384

[861] See online at <https://dl.acm.org/results.cfm?query=continuous_delivery&Go.x=0&Go.y=0> (retrieved July 29, 2019); see also above, N 182.

[862] Transcript M, N 13.

[863] Two companies emphasized that they also work with short-time releases. However, they would try to limit the releasing to a smaller number a year as every release would also carry the risk of a system breakdown which could not guarantee the stability needed in their business class (see transcript I, N 34, and transcript H, N 15).

[864] Transcript A, N 14, and transcript K, N 18.

[865] Transcript M, N 13.

[866] Transcript G, N 25, and transcript F, N 17.

grated at certain milestones of a project, for example to ensure that no intellectual property of third parties is infringed or to draft a contract if a collaboration with third parties is planned. The standardized development processes empower the engineer to work freely within their field of duty so that they know at which point other parties within the company should be involved. From the perspective of the company, standardized projects also help to mitigate compliance risks.[867]

385 The interviewees described how, nowadays, collaborations with *external partners* are formed more frequently in the industry. Companies often work with other engineering teams or tend to outsource particular parts of a project.[868] Big companies whose core business does not lie within engineering either open a separate developing group somewhere abroad or pass on the engineering or coding step to third parties to let them finalize their projects. This means that software development within a company often does not work like a closed shop but instead partnerships are made for selective collaboration. Collaboration partners in this context are, for example, suppliers, consultants or even clients who help with realizing and commercializing a project. Another popular method is to buy standardized software from a third party and adapt it in-house to the company's particular needs.[869] The selection of a suitable approach for a company is often related to a cost-benefit analysis and is dependent on the knowledge and know-how available within the particular company.[870]

386 To conclude, an evaluation of the software development process has shown that the previous approach of linear software engineering has been partly replaced with spiral methods or a combination of both methods. In spiral approaches we can still observe reoccurring phases of ideation, conceptualization and realization, followed by the implementing and testing of the created computer programs. The approach used for development is commonly decided on a case-by-case basis by the developer in smaller firms, while bigger companies tend to work with standardized engineering processes which define in detail which methods will be used in which situations and at which points further internal resources and external collaborations may be requested.

867 See transcript C, N 15-20.

868 See, for example, transcript F, N 17.

869 Transcript A, N 12.

870 Transcript F, N 29, and transcript E, N 23.

2. Duration

The interviewees were asked how long the development of a new software product or service takes on average. According to these companies, the time between a new project idea and its finalization can vary from a few months to several years. The longest period specified was nine years, which was needed for the development of a large-scale project, including a complete new system environment.[871] 387

The exact time needed to develop a new software product was said to depend on the particular circumstances and settings of a project. According to the interviewees, there are several factors that influence the total duration of the development process: 388

- complexity of a problem;
- conciseness of a client order;
- creativity and commitment of the involved engineer(s);
- innovation degree and novelty of the project;
- know-how and knowledge of the engineer(s);
- available resources (money, time, staff);
- pre-existing, reuseable software components;
- degree of engineering documentation needed;
- requested sustainability of the software system;
- administrative processes;
- decision-making processes of involved parties such as management or public authorities;
- expectations on the time-to-market;
- compliance issues (especially product clearing); and
- legal requirements.

Although there are many factors that have an effect on the duration, those with the greatest influence were said to be the complexity of a problem and the administrative issues connected with a project. At the same time, the great diversity of factors shows that the duration may strongly differ from project to project and that it is difficult to determine a reliable figure. Therefore, although we cannot make a generally valid statement for each case, we can find an average to work with in the present study. 389

[871] See transcript L, N 25.

390 From the small sample gathered, an average duration of the core development process (from idea to the first release of a complete software version) of approximately 3 years can be estimated.[872]

391 With the iterative approach, possibly combined with a continuous delivery approach, it is often difficult to tell exactly when a first version of the software product can be considered as "finished". Talking about the minimum viable product, the interviewees explained that in practice it takes an average of 2 years to complete a first presentable or releasable function of a feature, which itself may consist of multiple individual deliverables or increments.[873] Once the minimum viable product is formed, several two to four week sprints follow, in each of which a new function or feature is implemented.[874]

392 Although the small sample of 12 interviews does not allow statistical final statements to be made, it gives an idea of how long the development process might take. According to this, a linearly developed project may require an average of 3 years from its idea until its final implementation. Where the iterative approach is used, a period of 2 years, on average, may be needed to create a minimum viable product.

C. Programming in Particular

393 In order to determine how far protection should reach, we took a closer look at the engineering process. The interviewees were asked how and why a particular programming language was selected for a software project, to what extent they could rely on existing code and how easy it was to reproduce technical components, if required.

1. Selection of a Programming Language

394 The interviewees agreed that the developer can often solve the same technical problem with several computer languages. *This means that they are able to ful-*

872 Where the software company worked with the continuous release approach, the first major release was considered as a reference point to determine the duration of the core development phase. This point then also functioned as the starting point for the next delivery project. As the first release usually represents a big launch, the subsequent deliveries occur at shorter intervals.

873 See for example transcript A, N 30.

874 Transcript A, N 162.

fil identical requirements and display the same characteristics in different computer languages. The engineer can therefore choose between a set of languages and decide which one he or she wants to pick to realize the project.

Four factors were described that affect the selection of a programming language: 395

- the pre-existing technical environment or system;
- available engineering skills;
- trends and support of the community;
- the personal preference of the engineer.

The *pre-existing technical environment* appears to have the greatest influence on the selection. Several interviewees described software as working like an *ecosystem*; there are numerous programming languages that show a similar set of components and frameworks. This means that they are based on similar characteristics or principles and are therefore alike. As they are built on the same principles, they can also co-operate, and become combinable and exchangeable. Within the ecosystem there are also coding libraries, tools and computer-assisting programs available.[875] If a new element has to be integrated into an existing system or used on a specific platform, the characteristics of the program have to be compatible with its future ecosystem, otherwise the program and its future system will not be able to interact.[876] Software engineers refer to this as *connectivity*. Certain elements are dependent on others. The smaller the set of available programming languages within the same ecosystem, the more restricted the discretion of the software developer. One particular field of application is the market of smartphone app programming. Here the selection of a programming language is restricted by the two main operating systems available (iOS from Apple and Android from Google). As both producers provide a very limited number of programming languages each, the developer is bound by these few languages if they want to offer an app in the relevant store. Selecting a computer language thus can be *platform-related* as the developer is dependent on the software's future ecosystem.[877] The programming language therefore is partially predetermined by the subject you want to code and the environment you want to integrate your program into. 396

875 Transcript E, N 34, transcript I, N 51, and transcript K, N 28.

876 See, for example, transcript A, N 40 and 42, and transcript E, N 36.

877 For this problem see transcript K, N 26.

397 Another aspect influencing the selection of a programming language is the *skills that are available within the software developing company*. As with common linguistic languages within any other kind of business, engineers do not "speak" all computer languages.[878] Usually, they have a certain set of language skills they offer. Dependent on the engineer's particular language skills, a software developing company will tend to code in one type of programming language or another.[879]

398 As will be described later, there is a particular culture in the software engineering industry that supports the exchange of expertise and know-how.[880] As a consequence, one major criterion affecting the selection of a programming language is the predominant *trends of the industry*. Since the rise of the Open Source community and the provision of libraries and computer-assisting tools, engineers have been *supported by their IT-community*.[881] For example, the language C was very popular until the early nineties, but with the increasing importance of the IT community, Java has widely replaced C in most engineering sectors.[882] Now, if a developer has to solve a complex problem they may want to rely on pre-existing tools and build on solutions they can find in libraries or within the Open Source community. Consequently, if they do not use a widespread programming language the developer has to code everything by themself, which requires more resources. Engineers therefore want to work with a language that is well-supported by the community. The available components often show a high technical quality. This may support the longevity of the program it is implemented in as well as its persistence.[883] Similar to the analogue life, trends and the mainstream change over time.[884] The engineer therefore has to keep up with these changes and even sometimes learn a new computer language.

399 As with the selection of a linear or spiral approach, in many companies, engineering teams can select for themselves which programming language they want to use. If not restricted by a certain system environment or trend, the selection becomes a question of *personal taste*. He or she can "*choose the*

878 Explained illustratively in transcript I, N 51.
879 See for example transcript H, N 29.
880 See N 449 ff.
881 See explanation in transcript K, N 26.
882 This example was brought up by several parties. See for example transcript I, N 52.
883 Transcript K, N 26.
884 Transcript I, N 53.

means which suit them best".[885] As one interviewee explained, every engineer has certain preferences for particular languages.[886] If there is a set of languages they can select from, they usually take the one that in their personal experience has been proved to solve a particular category of problem best or easiest.[887] The preference for a particular language may also be due to a particular image the language has or the personal interest of the engineer. If they are a supporter of the Open Source community they are more likely to work with Linux than with Apple or Microsoft products, as Linux itself is Open Source and built on free software, while other distributors tend to be more proprietary. The personal background of an engineer may thus be seen in the component he or she produces.

To sum up, four main criteria were observed that can influence the selection 400
of a computer language in a software development project. While the particular requirements of a software system seem to be the main reason for selection, other reasons such as available skills, trends and support in the IT community as well as the preference of the engineer can lead to a particular decision.

2. Adaptability of Computer Programs

As seen in the previous section, the selection of a programming language is 401
rooted in four main criteria. What was further discovered in the interviews was that a programming language can be extended, combined with or substituted entirely by another programming language. Translation from one to another is possible as long as two languages from the same set or ecosystem are selected. Different programming languages offer different illustrations or expressions of the same steering algorithm. It is thus possible to exchange a programming language and translate the programming languages from one into another. This is supported by the fact that, like known translating services for linguistic languages such as GoogleTranslate and Skype Translator, there are also translators available that translate the source code from one computer language into another.[888]

The determining factor in translating a computer program is effort; changes 402
are always related to effort and therefore have to be backed by a cost–benefit

885 Transcript H, N 27.

886 Transcript H, N 27.

887 Transcript I, N 51 and 53.

888 See discussion in transcript A, N 44 ff.

analysis. The engineer has to work within the same ecosystem and follow its coding patterns and best practices. It is not just the translation that requires effort, but also making all the smaller adaptions so that the software is not only compliant but appears the same to the user. The more connections that exist in a software system, the more complex it gets to translate all the links and transform an existing code into a new language. Therefore, an engineer will try to stick to the old language or choose one that is compatible. Some companies have to change their source code from one language to another in order to increase the efficiency of their system, open up to new functionalities or because the former programming language is no longer supported by the community. At the same time, switching to another programming language always carries risks, as mistakes in translation can happen easily and the reliability of a system might suffer. Changing or altering code therefore usually requires a large amount of effort. Interestingly, the interviewees felt that if the reproduction of code took a lot of effort, this was the best technological measure to protect them against copying.[889] As copying the original already takes a lot of effort, even more is required to conceal the pilfering. However, as long as you comply with the outlined rules, the source code can be translated from one programming language into another. The effort needed to do so widely depends on the individual case.

403 What is true for code translation is also true for adaptations in the structure of a source code within the same programming language, known as *refactoring*. Overall, rearranging parts of code within the source code is possible but, depending on the available resources, a change in structure may represent tremendous expenditure. Again, a company might want to do this if a cost–benefit analysis predicts benefits, such as efficiency or readability. Or as one of the interviewees explained: "It is like a house: if you only want to alter the interior design, it will be cheaper. If you want to eliminate a wall, this will cost more. If you want to reconstruct the basement as well, this will become very expensive. And whenever you are attached to other interfaces or to a technology, you will face more problems and costs".[890] In general, however, rearranging the structure of the source code is feasible.

404 New technical solutions have made it easier to copy, paste and adapt technical solutions in computer programs. One can simply choose an element and integrate it into a new environment. Although it was clear for all companies interviewed that a source code can be altered and pasted, many of them empha-

889 Transcript I, N 22, and transcript B, N 174.

890 Transcript A, N 52.

sized that not every similarity or translation in a source code should be open to legal prosecution by the author.[891] If a certain amount of work has been put into the manufacturing of new code, this work should be valued too. The effort needed to create a new work therefore should be a determining factor for the evaluation of derivative- works, including source code copying.

To conclude, project elements can often be displayed in a number of similar programming languages. Within the same ecosystem, the developer is free to partially or entirely translate or adapt the source code. Blocks of code can be moved around and rearranged within the same source code. Both changes – the translation and adaption of the original source code – are not a question of possibilities but rather of expense. 405

II. Software Commercialization

The basic rules for trading are set in the legal system. Commercialization is an important issue for right holders as this is the way they can earn back their investment. What the commercializing strategy looks like varies largely from one company to another and may also change over time. This section first looks at the interviewees' impressions of the software market in order to then explore the different distribution models for software products, enlarging on the Open Source model in particular. It closes with a look at the average product life cycle of software. 406

A. The Software Market

There are several multi-state treaties that, at least partly, cover the protection of computer programs on an international scale. In the interviews, the parties were asked what significance the international market has for their work and whether they experience competition in the IT market. 407

1. Offering on the International Market

The market in software engineering was described as two-fold. Companies strongly differentiate between software they purchase from others to implement or integrate as demanders (consumer) and software which they offer themselves on the market (provider). Analysis of the interviews showed a strong tendency for software engineering companies to work internationally 408

[891] See for example transcript H 114 and 118, and transcript K, N 31.

in both perspectives, as consumers and providers. Many companies have obtained software from a supplier in another country, at least once, because either they did not have the skills to develop this product themselves or because the investment would not have been financially feasible. Most companies do offer at least some of their products on the international market. Europe, the United States and Asia were stated as the most promising markets.[892] Expanding to other national markets, as one interviewee explained, was easier in the field of software engineering compared to other industries, as the core of the product offered remained the same in every country.[893] By providing manuals and user interfaces in multiple languages, one can gain access to several million new potential customers worldwide. In addition, it was noted that the investment needed to enter the first national market is the highest, reducing for every new market conquered afterwards. Commercializing products internationally thus gradually reduces the economies of scale.[894]

409 There were some participants in the interview series who said they had not expanded to other markets, instead focusing on their home country's market.[895] These companies had in common that they either provided a very specialized service in a niche area or were in some way aligning their software product with particular legal settings that were relevant for the business sector they were supplying. This might be, for example, data protection law, IT security laws or public law regulations for a specific sector (telecommunication) or topic (tax and accounting). Supplying a business sector with a high regulatory density may therefore result in a company working less internationally as the expense of adapting the software system to a new legal environment would be too high.[896] This corresponds to the fact that the companies stated it would be a big disadvantage to expand into new markets as the legal systems were so different between different jurisdictions. This would cause great legal uncertainties. It was noted more than once in the interviews that the U.S. market was difficult to become established in because of its non-transparent and non-foreseeable court practice. Punitive damages in particular were said to increase the risk in offering products in the United States.[897] A

892 See transcript G, N 75

893 See transcript I, N 92.

894 Transcript K, N 55.

895 Some of these companies stated that they purchased software from foreign providers and that, to this extent, considered software engineering as an international business (see for example transcript I, N 92).

896 See for example transcript L, N 57.

897 See for example transcript C, N 54.

rational risk assessment would therefore often dissuade expansion into the U.S. market. As well as differences in jurisdiction, the legal systems, as observed in the patenting of software, were said to require a great deal of effort to analyse and organize commercialization strategies. Due to the extent of research and preparatory work, many companies desisted from expanding.[898]

To conclude, it was observed that the software engineering market in general tends to work internationally. Only a few companies limited their services to one country alone. The ones that did so had in common that they either provided a very specialized service to their clients or the software they supplied was used in a business area accompanied by high density of legal regulations. Differences in legal systems were described as a risk factor in expanding to further national markets and therefore a negative factor. 410

2. Free Competition

The argument of free competition is often brought up in response to an intensified intellectual property system, and not solely in the field of software protection. The question therefore is, what significance does competition have for software engineering as such and for a system including IP rights in particular. During the interviews, the questions on this subject were rearranged and reformulated several times. One problem was the difficulty of capturing the issue in one comprehensible question. At the same time, the question had to be open enough to not appear leading. I therefore asked the parties in quite general terms about the importance of free competition for software engineering. This meant that the interviewees could choose which aspects they wanted to focus on. The question could either go in the direction of what significance free competition had on their occupation, or lead to a political debate on whether competition was possible at all in a system with intellectual property law. The formulation of the question was successful in that I received feedback on both avenues with a wide spectrum of feedback and the latter seemed to be hitting a weak spot. 411

In the first answer category interviewees discussed what *significance free competition had on the occupation of software engineering*. I was here addressing the potential effects of competition on the trade and business of software development. All agreed that competition as such was something important 412

[898] Especially for smaller companies it remains difficult to evaluate the foreign market, and essential resources for the analysis of foreign legal systems are rarely available. See transcript K, N 55.

for producing and consuming software. Several interviewees felt that the software market did not differ very much from other industries and that competition is valuable for every market.[899] They described free competition as "motivating", "inspiring" and "empowering" for the participating companies,[900] explaining that in monopoly markets the parties would become "bored", "lazy" and "uncreative" and that developing in such an environment would not be constructive for a company.[901] Most of the companies therefore seemed to fully appreciate competition in their occupation, as it drove their work and they had to stay ahead of competitors and offer high-quality services and goods to maintain their customer demand. Competition would also help to guarantee a standardization process in technology, as competitors are able to learn from the practical solutions of others.[902] Competition was therefore regarded as a very positive thing, contributing to a good final product.

413 In the second strand, answers were expected on how free a market should be and *how the engineers truly experienced the software engineering market*. Companies here distinguished between software they purchased and used in-house, and software components or single programs they bought from third parties to integrate into their own software and then sell to their customers.

414 On the consuming side, the companies confirmed that they observed competition between sellers and that they were completely free to decide which company to buy from. They selected based on various factors and the role of the market in terms of price, performance, offered services, and so on.[903] Ultimately, the clients chose software based on their preferences, making supply and demand the key factors in their decision. The more specialized the product or service and the more dense the market becomes, the fewer the competitors available to choose from.[904] But even in these limited markets, competition can still unfold and the companies can decide between a smaller

899 See particularly transcript C, N 50.

900 Transcript C, N 48.

901 Transcript C, N 48 ff.

902 One interviewee in this context mentioned the term "copy with pride". He noted that keeping an eye on the market and adapting certain elements would keep the market creative and help keep it alive. At the same time, he emphasized that simple copies of a component were not helpful, as they did not make a constructive contribution to the market. For the full discussion, see transcript K, N 53.

903 See transcript L, N 55.

904 Transcript L, N 55.

number of specialized competitors. One may therefore conclude that free competition is warranted where companies are looking for a complete software solution.

It gets more complicated analysing free competition from the developer perspective where companies are looking for components and single programs they can implement in their own products. It was expected that interviewees would discuss the effects of exclusive intellectual property rights on the market, and they were unanimous that it presented a challenge for companies to develop within a market that is partially restricted by the intellectual property rights of other companies.[905] Further, they stated that too many exclusive rights for basic inventions could have negative effects on the market. Apart from that, however, the feedback was divided between two political camps; a bigger proportion of the companies interviewed saw exclusive intellectual property rights (particularly patents) as a reasonable institute in software engineering that had an effect on the market, but one that could be handled. They said that exclusive rights, even absolute rights to a certain extent would help to stave off the competition for a limited period in order to get established in the market. This would help companies to regain their investments and would consequently be a positive measure. A smaller number of companies, mainly the smaller or middle-sized companies, had a strong aversion to monopoly-like rights and emphasized that patent law in particular had a negative effect on competition, restricting the market dramatically.[906] IP rights were considered as strongly hindering, and even blocking, software development. I was struck by the clearly negative feedback from the smaller companies and was curious about the strong emotions I sensed in the answers given, which I could not fully classify. After the first couple of interviews I therefore added an additional question to the interview schedule. Regardless of the interviewee's answer to the first question, I asked whether and to what degree they had been hindered practically by an exclusionary right of a third party in developing their software. All but one interviewee stated that they had never themselves been prevented from developing software. Several had experi- 415

[905] Many companies explained that they had implemented a control mechanism or standardized processes in both buying and developing software to check that no intellectual property rights of third parties were infringed. See for example transcript B, N 60, and transcript C, N 52.

[906] Transcript H, N 126, transcript E, N 88, and part of transcript A, N 96.

enced impediments in one way or another, but all the companies involved could either avoid implementing the particular product or could switch to a similar service provided by a third party.[907]

416 What we can observe here is a clear gap in some of the statements; on the one hand, these interviewees emphasized that exclusive intellectual property rights, and patents in particular, would have strong negative effects on the market as they would hinder or even block others. On the other hand, none of them had been, de facto, hindered in developing a software product, which would appear to be contradictory. I see two possible explanations for this; first, as the interviewees had never personally experienced a blockage, they may have adopted their opinion on IP rights from a third party, such as a developer friend or the media, and had generalized their statements. Another explanation could be that the statements reflected fears of the interviewees that their company could be hit by the potential negative effects of patenting in the future. As the smaller companies noted, they had probably infringed someone else's IP rights by accident before. Their reaction in the interview might therefore represent anticipation of what they would feel if they were actually confronted with an IP infringement. Either way, we know that the involved companies have never been prevented from developing a software product, so competition in these cases might have been partially impeded but never fully eliminated in the process.

417 Another aspect discussed by some parties was the impact of licensing and the scale of the fees in particular. This was seen as the second potential negative side of an exclusive IP right in terms of competition. Indeed, many representatives of smaller companies described the pricing in the software industry as inappropriate and "harsh". The smaller companies particularly felt that licensing fees were too expensive and the terms too unilateral. Only lower fees would be able to foster competition in the market. However, a representative of the opposite point of view commented: "Free competition – that sounds okay to me. But free competition cannot become synonymous with 'free-of-charge'. Free competition for me means unobstructed competition; that nobody tries to act unfairly and can put obstacles in somebody else's way. Competition law has to answer this problem. But free competition does not mean that everything is cost-free. Because after all, big companies (...) have to pay the wages of their employees and taxes and they have to make investments.

[907] In transcript A, N 96, one interviewee talks about a specific feature the company wanted to implement but could not because of a standing third party patent. They then had to fall back on an Open Source product that offered a similar solution.

And if everything would need to be available at no cost, this would not work. I believe these issues get mixed up (...)."[908] This interviewee differentiated between the competition that has to be provided and granted in the market in general, and the desire of certain engineers for IT services to be provided at lower or no cost. I have noticed that personal interests and stereotypes often affect the debate of 'intellectual property rights vs. free competition', and that the various layers behind the problem are difficult to properly distinguish. The discussion became very emotional, making it hard to evaluate neutrally whether or not IP rights have a negative effect on the market. However, as no interviewees had been seriously hindered by a competitor, with alternatives always available, it can be assumed that "patent wars" are not a problem of daily business and rather become relevant in more established markets, if a company is big enough to raise the attention of others.[909]

B. Distribution Models

By using adequate and tactical price models, commercialization can be adapted to the practices of an industry as well as to the needs of a company and its clients. I asked the interviewees which models they used to commercialize their products. 418

Overall, they tended to work either with licensing or software as-a-service models. Companies in the mobile app sector provide their programs partially for free by cross-financing their products with others.[910] One company currently makes use of a 'freemium' model where the customers do not need to purchase the app but instead buy supplementary extensions if they want to benefit from a particular additional service. 419

On the whole, a clear shifting *trend away from licence models towards software as-a-service* was observed. Companies that continue working with licence models frequently charge their clients per activated user or per connected electronic device. Services, often development or maintenance services, are usually billed by the hour[911] or by registered access of a service on the server of the right-holding company[912]. The drift towards software as-a-service is partly associated with another trend, described in the section on selling elec- 420

908 See the original transcript in German: transcript G, N 70.

909 Transcript F, N 51.

910 Transcript A, N 20.

911 Transcript C, N 11.

912 Transcript L, N 12.

tronic hardware devices. According to the interviewees, until very recently users generally had to purchase the hardware products, while the software was implemented for free. Today, more companies are providing hardware components at lower costs or even for free, but instead will charge for the software products and services.[913] Recurring price models encourage a continuous relationship with clients instead of a one-time purchase. As a consequence, the software companies working with software as-a-service can make more profit with longer-lasting customer relationships and higher economies of scale[914] – two important factors that make software as-a-service a very attractive pricing model.

C. Excursus: The Significance of Open Source for Software Engineering

421 With the rise of the Open Source community and the Free Software movement, sharing working products and ideas freely online has been described as the most important trend in software engineering. Although the sharing community does not have a direct impact on the protection scope of software, it should nevertheless be recognized as an important factor in software development and commercialization as well as in copyright protection of software. Therefore, during the interviews, I took the opportunity to ask the participants about the significance of the Open Source and Free Software communities to their daily work and what are the major pros and cons.

422 The companies interviewed unanimously confirmed that the value of Open Source, especially, is immense and that the significance for software engineers is clear. Many stated that without the community it would not have been possible to realize their software or, if at all, only with the help of large third-party finance aid. They stated that a major part, sometimes up to 50 per cent, of the source code is based on programs and code provided by third parties such as the Open Source community.[915] Without this opportunity, the required workload would increase substantially. The available selection of libraries, source code, programs, routines, loops, frameworks, even whole computer operating systems such as Android and Linux, is large and the benefits of using Open Source programs and Free Software are said to be outstanding. All of the companies had integrated at least some parts of Open Source into their software.

913 See transcript G, N 15.

914 Transcript G, N 15; see also transcript L, N 14.

915 See for example transcript I, N 82, and transcript K, N 47.

Several stated that only a very small percentage of the software on the market had not integrated online-available third-party components.[916] Others have claimed that without Open Source software engineering and maybe even technology standards in general would be lagging behind for up to eight years.[917] They describe how although Open Source and Free Software are important, these models could perfectly co-exist with and complement the classic proprietary solutions.[918]

While many consume from the Open Source community, very few of the interviewed companies shared their own technical solutions or made them accessible. The know-how necessary to develop software can be seen in the source code.[919] If a company provides an excerpt of their software online, and thus offers insight into their technical solution, they could endanger their proprietary distribution models. Why should any customer pay for software that is partially available for free? The use of Open Source in this case is deliberately one-sided to prevent know-how from leaving the company. Others argue that they would be willing to share their work, but they consider their developments to be too specific, and the community would not profit in any way from them.[920] If a technical solution was designed to work only in a very specific field of application, it would not be transferable to another one. Providing computer programs within the community, hence, requires *a standardization effort* to make it useful and relevant to others. Most companies do not have the resources or incentive to contribute and make this effort, although they would like to.[921] In order to offer a minimal contribution, companies therefore often allow their employees to code for the community and start their own small projects in their leisure time.[922] Active participation, however, is still limited to bigger companies with more resources and a relevant business strategy.[923] 423

916 See for example transcript F, N 41.

917 See for example transcript A, N 80.

918 See the same conclusion in transcript M, N 29.

919 For more information regarding the visibility of know-how in software components, see N 441 ff.

920 See transcript K, N 51.

921 Transcript I, N 86.

922 See for example transcript B, N 54, and transcript A, N 88.

923 One example of a company providing code within the community is found in transcript F, N 45 and 73.

1. Chances and Benefits

424 The emergence and acceptance of Open Source and Free Software is closely related to the benefits it brings. The main advantages are its accessibility and quality, its reference to a community spirit, that it is cost-free and that it supports technological standardization.

425 *Accessibility* – with the rise of the Internet, the exchange of know-how has increased dramatically. As the Internet stores the provided information and elaborated solutions, the engineer can select from a mass of accessible libraries, forums and platforms. This makes software engineering faster and simpler, and lowers search costs, as the individual engineer does not need to develop a solution to every problem, but instead can rely on pre-existing products.

426 *Quality* – Open Source software usually implies good quality. As many solutions for similar problems are offered, usually only the strongest and best prevail. The products are consistently reviewed, tested and enhanced by experienced members of a huge community. Therefore, bugs, weak spots, even security holes in a program are revealed more easily. As it is a characteristic of the community to continuously improve a product, the discovered problems can be fixed quickly and simply.

427 *Community Spirit* – as will be explained later,[924] in the technical questions especially, the interviewees described how engineers have a strong wish to exchange their know-how. Open Source platforms provide them with the base they need to do so. It "gives engineers the possibility to learn fast and develop fast".[925] The community thus enables the industry – similarly to the function of patents – to learn from each other and enhance the state of knowledge by offering solutions publicly. Another feature of the community spirit is the openness of the system; everyone is able to participate by consuming or providing content. It is then up to the community which solutions are accepted and enhanced. This is part of the "democratization"[926] within the community.

428 *Cost-Free* – smaller companies have very limited time and financial resources. They are grateful that they do not have to discover every problem's solution

924 See N 449 ff.

925 Transcript A, N 80.

926 Transcript H, N 63.

themselves but, instead, can use available components from others entirely free. Considering the established licence models in software engineering, Open Source therefore offers a cost-efficient alternative to licensing fees.

Standardization – it could be related to the described democratization within the engineering community that certain trends are formed within the community. As the expert community decides for itself which solutions are the best and therefore which to preserve, a standardization process automatically occurs. As explained before,[927] software engineers' companies like to use technical solutions, for example programming languages, which are well-supported by the community. The whole community therefore is pooled and moves together in a specific, unified direction. This causes a standardization movement in which the whole community aligns, and adjusts their approaches and solutions. 429

2. Risks and Negative Effects

Although the positive effects of Open Source are evident, the interview series showed that while most companies make use of provided content, only very few also share their content and know-how with other engineering companies. The reason for this is that there are certain negative aspects and risks in using Open Source software. 430

There are many different Open Source and Free Software *licence types* available which each have very different characteristics. This makes it hard for every developer using Open Source to oversee the terms and conditions of the integrated components. Similar to a proprietary licence contract, there are certain requirements a licensor has to meet in order to receive a contractual right to embed Open Source or Free Software components. The most restricting type of licence is probably a Free Software licence called GPL (General Public Licence), as it requires the licensor to open his or her whole source code to the community (called back-licensing). Using code that relies on a GPL therefore affects, or "contaminates", the rest of a computer program, if it is distributed.[928] This makes GPL unattractive to those companies that are either unwilling or legally restricted from licensing their software back to the community. These restrictions may, on the one hand, be due to contractual agreements with clients who paid a developing firm a certain sum for an exclusive 431

927 See N 398.

928 For this reason, Open Source and Free Software is sometimes referred to as "contagious"; see particularly transcript G, N 58 ff, and transcript C, N 32.

development and now do not want the software to be shared with third parties for free. But the restriction may also be rooted in the need to protect sensitive (personal) data.[929] Companies can be forbidden from opening up their source code by law, for example due to IT security concerns or telecommunication statutes. If companies work with some kind of sensitive information, they hence may, de facto, be hindered in using Free Software or stricter Open Source licences. In this case, particular content that is constrained with a strict licence type in software developing companies usually has to be authorized by the legal counsel or avoided entirely.[930] Instead, the companies try to provide their engineers with tools and libraries they can use without problems. Some interviewees also referred to a particular software their companies used to scan the source code of their products and services in order to detect whether a Open Source component was integrated.[931] Other developers are not equally concerned with the problem, as their companies do not sell the developed products on the market but rather make use of the software in-house. As they do not distribute their developments, they usually do not come under the terms of the GPL licence. Also where companies sell individual software to specific scattered customers, the risk that anybody will ever analyse their source code and determine an actual infringement of an Open Source or Free Software licence is quite small, even though it cannot be entirely ruled out. There are many different ways in which companies react to the challenges occurring with the use of Open Source software. Clear guidance for the engineering departments with a comprehensive policy, was particularly noted to minimize the potential risk of using online components.

432 Another aspect mentioned was that, similar to 'common' software, Open Source and Free Software, too, may contain bugs and security holes. But as the components are made available to a larger audience, erroneous solutions are spread on a larger scale and have a quantitatively greater impact. It is probable that a considerable proportion of the community is "*affected by the same infection*".[932] As so many engineers or developments are affected, errors in Open Source solutions may become a question of unknown liability. As described by

929 See particularly transcript F, N 73.

930 See for example transcript H, N 65.

931 One supplier who was mentioned several times in the interviews is Black Duck (<https://www.blackducksoftware.com>).

932 One described the effects of the so-called 'Heartbleed Bug' in 2014; an error in a security protocol provided by the Open Source library meant it did not encrypt data the way it was supposed to do and the personal data of users could therefore be extracted more easily. Discussed in transcript K, N 49.

some developers, often certain limits to the warranties in Open Source licences are set that restrict the users from integrating Open Source software in high-risk environments, such as steering systems of planes and atomic power plants.[933]

3. Closing Commentary

Several interviewees stressed that working with Open Source does not mean that companies can no longer be profitable. Larger companies try to integrate the Open Source model into their proprietary models. This happens partly because Open Source offers some convincing benefits, mostly however it is done because the users and younger engineers are demanding that bigger companies adapt and rethink.[934] One company that was described as following this dual-licensing model, where Open Source and classical proprietary licences are combined, was the U.S. company Red Hat. Calling itself a leader in providing Open Source software,[935] Red Hat offers a wide range of Open Source programs and component licences. Most of its offers meet the essential needs of many engineers and are thus highly standardized. Additional specialized services and offers with adaptations are licensed in proprietary models for a fee. Through this, Red Hat has built its name, making it more reliable for engineers and offering tailored solutions for companies requiring more enhanced technical solutions.[936] This example shows how software engineering companies can adapt to new properties of the market and how remedy models might change over time.[937] 433

To conclude, this section has shown that Open Source and Free Software have a major significance for software engineering. Both may still be gaining in importance and continue to do so over the next couple of years. Content provided by the community is incorporated regularly into software and has become a necessity for software engineering because of its useful qualities. However, the interviewed companies, especially the legal counsels, stressed 434

933 Mentioned in transcript A, N 82.

934 See for example transcript M, N 29.

935 For more information, see <www.redhat.com>.

936 Red Hat was mentioned in several interviews, for example in transcript H, N 80. You can find more information about their services at <www.redhat.com>.

937 According to a press release on October 18, 2018, IBM is set to acquire Red Hat by December 2019 (see <https://www.redhat.com/de/about/press-releases/ibm-acquire-red-hat-completely-changing-cloud-landscape-and-becoming-world's-1-hybrid-cloud-provider> [retrieved September 6, 2021]).

the risks and challenges of using Open Source. Still, the benefits outweigh the risks, forcing companies to keep using them but limiting the negative effects with strict policies.

D. Product Life Cycle

435 An important question for my research was to figure out whether the current duration for which computer programs are protected under IP law is adequate. The main aspect of this problem was how long average products are in use. By estimating the approximate time a product can be distributed – the so-called life cycle – companies can calculate the period in which they have to work off their investment. I therefore asked the companies for an estimation of the length of the life cycle of their software products. To try and unify the results for the qualitative responses, we counted it from the moment when the software was first implemented to when it was terminated or had to be replaced with a new system or a larger "rework". It was difficult for the interviewees to determine these start and end points because testing and maintenance regularly distort clean cuts. Another difficulty was that most of the interviewed companies were currently offering several software products on the market, so they could only make an educated guess about an average runtime of all of them. In addition, most of the projects discussed were still live and running. The interviewees could consequently only predict an expected life cycle end. The feedback thus may be partly imprecise, but it still provided a lead on how long software products are commercialized in practice. As these numbers were based on the previous experience of the engineers, I considered them to be quite accurate.

436 Further, companies that work with the *continuous delivery model* do not think of software as a fixed subject that can be completed at some point. Rather, some of them expect, or at least hope, that their software system will be able to last almost indefinitely.[938] As the developers can react to difficulties, they are able to meet the users' needs more flexibly and at the same time maintain their software technically. The continuous delivery approach may thus be able to prolong a program's life cycle. For programs built on the continuous delivery approach, we agreed to focus on the first basic software version and thus calculated from the program's initial implementation up to the point when the

[938] See for example transcript K, N 18. A practical example of long-running continuous delivery software is Google's Search Engine which has been in use since 1998 (see The History of SEO). Today, approximately 21 years later, Google is still offering this service and we do not expect it to be terminated in the near future.

system could no longer be re-engineered in further cycles or supported with additional add-ons. Alternatively, we could have concentrated on the separate modules, which are added individually, to calculate the life cycle. Either method of calculation is possible. For the present study, I decided to work with the first model for two reasons. First, the software as a complete product is a unit that can be separated more easily. Second, the interviewees described how today's users look at the full package, searching for computer programs that satisfy several needs at the same time. By implication, a single feature or component is not as valuable to them as the whole program. Therefore I wanted to focus on the whole program and not just one aspect. This also allowed me to obtain more detailed information about the sequence of the individual partial releases.

Overall, the software developers estimated the expected life cycle of a software product to be something between 5 and 25 years. As a rough average, the interviewees said that it takes 10 years before a system has to be fully replaced. Regrettably, some projects had to be abandoned after a shorter period of 18 to 24 months because the projected business cases did not work.[939] However, successful software products may go through several cycles of a complete software system if the demand is sustained:[940] They are fully maintained, newly integrated into a new system structure and afterwards released again after a total revision of their system. Where such a re-release occurs, the development project is assumed to require even more careful planning and farsighted anticipation. 437

The companies were also asked if they could explain which factors influenced a software life cycle's length and how they realized a new system had to be prepared for development. They said that to a certain degree software could be adapted, improved, enhanced and maintained within its own system. New code may be added, new system connections may be made, but after a certain amount of time, the effort or costs of maintaining the existing system become too high to continue. Or else the technology as a whole changes so much that the system has to be adapted to newer standards.[941] This is when the end of the life cycle is reached and the companies have to reinvest in new software – the so-called point of no return.[942] 438

939 See for example transcript B, N 16.

940 Transcript B, N 12.

941 Transcript L, N 16.

942 Transcript A, N 26, transcript E, N 19, and transcript I, N 34; in transcript I, N 36 the interviewee in this context spoke of the "costs to get off the technology".

439 According to the companies, the life cycle of software may be significantly extended by measures that keep the program manageable and transparent for the engineer working on it or increase the efficiency of the system. A clear structure and neat building of the code as well as good documentation of the engineering process were described as the minimum standards for a software developing company.

440 To sum up, the life cycle of a computer program was defined as the term between the software's implementation and the point when it has to be terminated or replaced with a new system. The software system has to be replaced when it becomes too expensive to maintain it. It was explained that the average duration of a software life cycle is around 10 years, depending on the particular circumstances and the preventive measures taken. New developing approaches make it more difficult to determine exact start and end points in a life cycle. The longest time period indicated was 25 years.

III. Know-how in Software Engineering

A. The Significance of Knowledge and Know-how

441 In order to optimize the legal protection of software, we have to find out where the (added) value of the developing company lies.

442 According to the interviewees, one of the most valuable factors in software development is a software developer's knowledge. They distinguished between three spheres of knowledge: a developer's specialist knowledge or *expertise*, his or her working experience, and the vertical domain know-how. The first term was commonly described as the technical knowledge of a person about the specialist's standards and principles. The interviewees explained that this knowledge was usually gained from an apprenticeship or higher education. The working *experience* of a software developer referred to the practical knowledge he or she gained working on the job. Experience would help a developer to evaluate what technical or procedural measures had proved to be useful or what to go without. And, finally, the *vertical domain know-how* referred to the in-depth knowledge of the particular business segment in question, the applied use, including particular knowledge about the business segment, its clients and their expectations, processes, structure and operations as well as the possession of valuable contacts. Following an illustration of one of the interviewees, the relation between the three types of knowledge can be

described as the following:[943] If one wants to build a concrete wall, the traditional specialist knowledge is the armouring irons that hold the concrete. Your experience tells you how to raise the wall, how to start and which instruments to use, but it is the domain know-how that tells you how the cement should be compounded and in which place it can be utilized most economically and efficiently. All three spheres of knowledge therefore interact. One cannot work without the other, but it is the exact combination that has an effect on the final product and service.

All the interviewees agreed that both experience and domain know-how are key factors of their work and that both enable them to distinguish themselves from their competitors. Although in the interviews they distinguished between experience and vertical domain know-how, they commonly used the term *know-how* if they wanted to include both at the same time. For them, the term know-how referred to the practical hands-on knowledge of an engineer. This know-how could either be based on long-term working experience or on particular knowledge of a business segment.[944] The interviewees described how a software developer's know-how influenced the development and resulting software product on several levels: 443

- in discovering and reacting faster to new trends;
- in how a project is developed and which highlights are made;
- in how collaborations with other partners are established and retained;
- in elaborating and specifying requirements in conceptualization;
- by knowing where the difficulties lie in a technical implementation;
- in talking effectively to clients in order to shape an order;
- by knowing what are the best practices of software engineering;
- in incorporating additional benefits;
- in drafting a user-friendly and handy design and framework;
- in choosing appropriate key interfaces;
- in structuring and documenting code;
- in developing creative solutions and programs;

943 The original explanation may be found in transcript G, N 46. For the illustration above, it was slightly adapted.

944 The two spheres were described as hardly distinguishable, if you have to determine in an individual case what sphere of knowledge was concerned. Hereinafter, I will therefore refer to both spheres of knowledge at the same time when speaking of know-how, without emphasizing a case in particular.

- by knowing what has been tried and tested and therefore is proven to be effective;
- in realizing projects faster and more effectively;
- in a faster reaction to possible problems;
- in evaluating the market;
- in commercializing software;
- in using resources efficiently;
- in anticipating likely problems;
- in knowing your clients, competitors and your resources; and
- in producing a program that can be monitored and adapted easily if needed.

444 Both experience and domain know-how have an impact on how an idea is realized technically. The interviewees believed both were important assets they had to protect. At the same time, the software companies highlighted that the know-how involved in creating a software product is often *visible* in the produced software component. As the accessible source code represents the commands the computer should execute, the developer's know-how invested in expressing these commands is displayed on the surface.

445 While a third party can acquire specialist expertise, the experience and domain know-how is something that evolves from within a company and is not accessible to just anyone. Further, the interviewed companies described how vertical domain know-how would be important for distinguishing a company's service in software development, operation and maintenance. Although there are certain niche areas, there are many software developers on the market who possess high-quality specialist knowledge of the technical aspects. Similarly, there are many developers available with long-term experience in the field. These experienced employees are free to change their work place and a company cannot stop them from using their experience. Therefore, companies try to stand out from the crowd by selling their know-how of a particular business segment instead, where less competitors are established. This is particularly effective for fields in which an in-depth understanding and know-how of the area in question is rare and where it is important to meet a customer's needs in technical implementation. Domain know-how has become very valuable for software developers because this may be the way to distinguish your business from other professional and experienced software developers. The interviewees highlighted that domain know-how was the biggest focus point when a new employee was trained on the job as it may help afterwards to com-

plete a job more effectively.[945] And it is also domain know-how that a company wants to retain legally when an employee leaves a company.[946] The domain know-how is said to represent the key asset for a software developing company.

Another interesting finding was that higher education in computer sciences was described as being similarly valuable to long-standing work experience in the field of an engineer from another technical physical science. Many of the people interviewed were originally physicians or engineers, as thirty years ago computer science was only just evolving. With the shift of the industry from mechanical regulators to software, many technical workers switched to computer technology. Today, the younger generations can benefit from higher education possibilities in computer science. They obtain the IT knowledge, the expertise, in their studies. Previous generations with education in non-digital subjects instead acquired their knowledge by experience.[947] From a sociological perspective, it is therefore interesting how the composition of knowledge in software development has changed from workers who develop based on their work experience to a combination of younger workers with higher education in computer science with the experience of long-term practising workers. As emphasized in many of the interviews, both experience and expertise knowledge is valuable and sought after equally in the ICT industry. The work experience of classic engineers, machine builders and similar professions may therefore compete with the specialist knowledge of newer software engineers obtained from higher education. 446

To conclude, the interviewed companies differentiated between three spheres of knowledge: specialist knowledge or expertise, a person's work experience and a person's particular know-how of the vertical domain in question. Although all three spheres of knowledge are recognized and described as important for software development, the know-how of a domain is critical for the competitive strength of a company and therefore represents the key value of a company. When computer science was still a new industry, the work experience of specialists in the physical sciences was most valued. The knowledge of computer science graduates is now able to compete with a worker's established experience. Consequently, both expertise and experience are seen as valuable and comparable in quality. 447

945 See transcript E, N 68.

946 See for example transcript E, N 68.

947 See for example transcript G, N 34.

B. Knowledge Transfer and Protection

448 As emphasized above, in all the interviews know-how was described as having significant value for the ICT industry. At the same time, the topic of know-how conservation is also gaining importance. The interviewees described how this topic currently represents a key asset in many software-developing companies. The question of how know-how is protected and how and why it is transferred has become an important subject for legal evaluation.

449 What we face in the engineering sector is a special 'culture' that is very keen to promote a voluntary exchange of know-how. At least, this is what is claimed regarding software engineering in general. I tried to detect whether those concerned shared this understanding of the software industry. The findings were split and rather complex in detail. As described above,[948] knowledge in our understanding may be differentiated into three spheres: specialist knowledge, experience and vertical domain know-how. What all the companies described was that in general the engineering business does indeed make use of platforms where an exchange of know-how is encouraged.[949] On closer analysis, however, it seems to be mostly the technical expertise and knowledge gained by experience that is exchanged between the engineers, while the domain know-how of companies is mostly kept secret. An exchange of the latter type of knowledge is limited to cooperation with selected partners.[950] The statement above was therefore only partially confirmed by my findings.

450 Software companies pursue knowledge exchange by visiting engineering conferences and using specialized platforms online (blogs, videos, forums etc.). Most of this exchange is made, as expected, within the Open Source community where full code lines are also made available. The Internet therefore has become the most important conservator and source of expertise and know-how based on experience. Vertical domain know-how, on the other hand, is usually not shared with third parties without a special reason. As more collaboration within the development and commercialization process occurs,[951] it becomes unavoidable to conduct a certain amount of exchange of domain know-how, at some point or other, in order to achieve a company's goals. Vertical domain know-how will then be exchanged, but it is commonly combined with a non-disclosure agreement or another form of legally binding contract.

948 See above N 442 f.

949 The most named sources were the Open Source community, forums and conferences.

950 See for example transcript K, N 41, where this tendency is described.

951 See N 385.

This is necessary because in this situation highly sensitive materials such as blueprints, instructions, workshop documentation and certain parts of technology are issued to the business partner.[952] All sensitive material that leaves the company and is shared regularly is protected with additional technical or legal measures. At the same time, all the companies agreed that there was no way to protect your goods with absolute certainty; if someone wants to get in, they will get in. In any case, software companies will try to protect their vertical domain know-how.

451 However, I discovered one interesting exception where domain know-how was exchanged more freely. It was noted that some companies do not restrict access to a technology or business model through technical or legal measures and instead make significant parts or even the whole software public. This practice was observed in cases where software companies wanted to propagate their development. For this purpose, a company provides more information to the background of their development or even distributes it free to use. The interviewee explaining this point illustrated his observation with Apple's way of commercializing their iPhone:[953] In order to get more suppliers to offer their products in the App Store, Apple has, so the interviewee explained, introduced their own programming language called Objective C and started offering training on it. Today, Apple also shares some of their business know-how with potential app developers and shows them how to optimize their apps. According to this observation, it would appear that companies such as Apple encourage third parties to make use of their development without requesting direct remuneration. A similar approach was followed by one of the interviewed companies. The interviewee explained that, in this way, know-how and essential information could be exchanged more easily and that the needs of involved parties, including the customers can be better integrated.[954]

452 Another observation was that to enhance know-how, smaller companies have to rely more on know-how exchange with third parties, while bigger companies can exploit more of the in-house potential because they have more engineers who can communicate and share their personal technical understanding. In the last scenario, some interviewees outlined that it would be a challenge to organize the available know-how into something useful by imple-

952 See transcript G, N 50.

953 See discussion in transcript I, N 72; the same issue was brought up in interview K, but off the record.

954 Transcript M, N 23.

menting frameworks as innovation workshops or think tanks.[955] Not only preserving and transferring, but also enhancing know-how therefore is important for companies.

453 In summary, know-how protection, transfer and refining is a big issue in every company. The industry of software engineering is said to be very open in terms of know-how exchange. A closer analysis instead shows that they do have an exchange-friendly culture but that it is mainly conducted in the spheres of specialist knowledge and experience. In the sphere of vertical domain know-how, the exchange tends to be limited to special situations and is often restricted through legal measures, as the vertical domain know-how is sensitive to attacks. An exchange of know-how is only conducted if it represents a major benefit that outweighs the risks of potential disclosure or abuse.

IV. Creativity and Innovation in Software Engineering

A. Creativity

1. What is Creativity in Software Engineering?

454 The main function of copyright is to offer legal protection for intellectual creations. However, the terms "creativity" and "creative" can have various meanings in different contexts. They may for example refer to the circumstance that a work was made, generated or developed. Where previously there was nothing, something has been constructed. At the same time it could also be suggested that a development result involves an artistic characteristic, that the product offers some original strength. The definition of creative working and its perception in the context of software engineering is vital in order to define what a copyrightable work should be. The software developing companies were thus asked what they understood by the term "creativity" and where they believed most creativity could be applied in their work.

455 The interviewees offered many definitions for the term "creativity". Surprisingly, they were all very similar. Summarizing from the interviews, the term creativity can be paraphrased as follows: *Creativity is the way a problem is solved and the way a potential solution is implemented.* As you start with nothing but a computer as your tool and end up with a (set of) computer program(s), the whole process may be understood as creative, because something

[955] See for example transcript F, N 29, transcript H, N 55, and transcript G, N 48.

new has been constructed. Even though the result remains virtual and is not physically tangible, something has been made.[956] On the other hand, the development process was also described as having an artistic nature as the author's personal skills and preferences, in their personality, will affect the outcome as well as its quality in various ways. Consequently, the interviewees saw software engineering as creative in both a constructive and artistic way.

456 It was described how there is *scope for creative decision-making in every development process*, and that it can be exploited in every phase of a process: in seeing a business need, in displaying these needs in the requirements and features, or by implementing a user-friendly graphic user interface. It can affect the way programming languages are selected and applied for coding. It can also influence the quality of the outcome, including its structure, readability and adaptability. The interviewees emphasized that there are no limits as to how creativity can be applied.[957]

457 However, not every software development project involves creativity in the same way. If the engineering process is limited to mere implementation, and does not build on creative leeway, creativity plays a very minor role in its production. If it is produced "*quick and dirty*", a program is made within a short time and without a lot of effort.[958] We should therefore see creativity if room is provided to make productive decisions and if the approach of the originator can have an impact. Creativity hence lies in a particular *contribution*. What this contribution looks like in practice may vary, making it important to look at the individual project: According to one interviewee, it is visible in the particular details of a product.[959] At the same time, the interviewees distinguished between using creativity from a business perspective and in implementing a product technically. The *business perspective* refers to how a project is built up. It is linked to the question of how a process can be shaped so that the business goal of an idea or need can be realized in an optimal way, such as which approach is selected so the users' preferences can be best met. The *technical implementation*, on the other hand, refers to the question of how particular challenges can be overcome technically, for example, how an engineer realizes a particular concept during coding. Here, functionality and efficiency criteria play a major role. At the same time, these aspects are said to become decreasingly important if data carriers happen to be more tolerable, store capacities

956 See transcript H, N 47, and transcript I, N 66.

957 See in particular transcript H, N 66.

958 See for example transcript H, N 47.

959 Transcript F, N 23.

increase or technical performance is optimized.[960] Still, from the engineer's perspective, a well-structured code is said to be helpful in enhancing and maintaining a program after its implementation. A manageable program facilitates its application by the user and, thus, increases his or her satisfaction. All of these decisions are considered to be creative in software engineering, as ingenious solutions for concrete problems have to be designed. While some decisions can be easily categorised into either the business or technical group, the composition of the look-and-feel for example has characteristics of both groups: It involves technical decisions to define requirements and features, but also refers to the original business idea by addressing particular needs and preferences of the user that will help in commercializing a product.[961] Creativity in software development therefore refers to the constructive process as well as to an artistic product. It can be exploited whenever a developer makes decisions that are relevant from a business or technical aspect. If the law wants to protect a work's creativity, this way of offering a creative contribution in software development should be considered.

2. The Impact of Toolboxes, Libraries and Assistance on Creativity

458 One assertion in the literature is that software engineering has lost its creativity as more and more engineers are relying on pre-existing components found in toolboxes, libraries and within the Open Source community. In the interviews, I brought up this claim and asked what impact the existence of templates has on the work of a software developer.

459 All of the interviewees confirmed that they occasionally use templates and source code snippets. Similar to online-shared know-how, these third-party components available within the IT community are of high value for engineers. As software engineering often involves solving similar problems in different contexts, the work of the developer is simplified if he or she can rely on previous works, either developed by the engineer or by a third party, because they save time by not 'reinventing the wheel'. Available tools and templates thus help to make the developer's work more efficient.[962] This is particularly important as the market of software engineering is picking up speed and, after all, 'time is money'! As discussed by one of the interviewees, ensuring data secu-

960 Explained several times off-tape but also mentioned in transcript G, N 44.

961 See for example transcript K, N 37.

962 See transcript A, N 54.

rity online with an encrypted function used to be a very complicated problem. The engineering process was complex. On the other hand, it is an issue every web shop provider faces and thus has to solve. Since the code functions for security certificates have been available in libraries, they have been easily embeddable by adding a particular PHP function to the program.[963] The accessibility of tools and libraries thus supports the developers in carrying out their work efficiently.

However, the other side of this is that, if a developer chooses to work with provided tools, he or she is de facto restricted from selecting his or her own solutions and instead has to work with the technical framework the tool offers. It is thus helpful to work with solutions that are well established in the IT community. Further, while the tools may facilitate some parts of the coding, the developer still has to engineer the rest of the program, embed it properly and connect the loose ends. Some interviewees explained that using tools can make their work more sophisticated; they have to keep up with the trends, continually acquire new skills, and develop the know-how about which tools to use in which situation. The developer has to acquire personal experience in how best to adapt pre-modelled solutions to a problem and integrate the foreign body into its new environment.[964] 460

The allegation that software engineering has become more uncreative in the last decade is partly understandable. However, the interviewees outlined how the selection of potentially suitable pieces remains a challenge in software engineering and still allows, and sometimes even requires, engineering creativity in order to offer satisfactory final products and services, including finding new solutions for which no model code exists. Even where some previously existing components can be used, specifications and adaptations to a particular system architecture have to be built individually. Where there is room to combine existing elements and tie them together, creative strength may be displayed. 461

B. Innovation

Innovation represents a competitive advantage for every company. At the same time, society can profit from innovations if they are shared. It is there- 462

[963] See for example transcript I, N 57; see also online-accessible instructions to implement a PHP function, for example at <http://skills2earn.blogspot.com/2011/03/how-to-encrypt-decrypt-url-or-code-in.html> (retrieved September 6, 2021).

[964] See for example transcript E, N 45 f.

fore in both interests if innovation is fostered. Patent law tries to do this by granting legal protection to innovation that is shared. It is widely disputed whether software as such is capable of being inventive. The term 'innovation' thus has to be used cautiously in software engineering compared with other areas of industrial innovation. In the interview series, this difficulty was tackled by asking the companies how they would define "innovation", how it is accomplished in software engineering and where in the development process innovation is located.

1. What is Innovation?

463 The interviewed parties were asked to define the term "innovation". When I started analysing the first few interviews, I realized that for many parties it was easier to describe what they considered to be lack of innovation rather than the opposite. In response to those first observations, I integrated further questions into the interview guidelines that covered trivial discoveries and qualitatively poor patent subjects. This step helped to guide the subsequent interviewees better to find a more precise definition for innovation. The result was that it also helped to delimit what could be excluded from the patent law's scope.

464 So what is innovation? Although most struggled with this question at first, to my surprise the feedback showed a clear common tendency. The shaping characteristics were the existing criteria for patents such as novelty and disclosure. But many also felt that, in addition to the existing criteria, innovations should have to show further qualities in order to be recognized as something worthy of protection. They would have to demonstrate an 'additional use' and outline the approach to the solution in a comprehensible way ('comprehensibility').

465 Taking all the answers into account, the term innovation can be summarized as: comprehensible added value for a third party that represents a novelty or includes progress from the initial position.

466 Apart from this progress or novelty, as mentioned, the interviewees wanted an innovation to offer *added value for third parties*. The companies expected a minimum positive effect or contribution – a benefit – that the invention has for society, the user or the market. This particular characteristic that the invention has to hold becomes more important if, in exchange, legal protection

is to be granted for the invention.[965] If an invention was only useful for the inventor alone and did not show a positive effect for others, it would not satisfy this criterion. This finding was surprising, as patentable inventions currently do not need to provide added value *per se*. The added value for third parties is shown in interest in the intellectual good, its worth to society. If the invention was only used to block others, there would not be any positive effect for third parties and it would not be deserving of protection. Well-established teachings have already been discovered and exploited. Society is thus less interested in granting them protection as their value is now limited for them. On the other hand, highly specialized knowledge that can only be used in its original context cannot be applied in another field. Again, its value for society is limited. In order to determine a model for protecting inventions, we should consequently focus on the type of inventive knowledge that offers a benefit for parties other than the originating one and whose application is not too specific or established in a particular area of practice. The interviewees also stressed that the value does not necessarily have to be an improvement in a solution's efficiency.[966] Instead, the benefit can be achieved by any essential improvement for a third party. The criterion can thus be regarded as flexible for a particular situation.

467 In order for an examiner as well as society to determine whether an invention offers added value for third parties, the interviewees felt that the technical solution should be disclosed in detail and described *comprehensibly*. Its core needs to be theoretically accessible to the market and its users. The invention should be described in a way that enables an evaluation of its added value.

468 With patents, the law wants to protect inventions legally. This definition of an invention exceeds the present, internationally agreed list of criteria of patent law for an invention, offering more facets. With an additional requirement, higher expectations of the inventive step would be set. This would make it more difficult for some developers to protect their inventions. On the other hand, the interviewees emphasized that asking for added value in inventions would positively influence society's perception of legal IP protection because protection would only be granted for what could add value to society. This might enrich the quality of newly developed inventions.

469 It was shown that the interviewees had a very distinct but unified understanding of the term innovation and an inventive contribution. They offered several

[965] See transcript B, N 34.

[966] See for example transcript E, N 54.

important factors to be taken into account for delimiting the innovation term. It was noted that the invention not only needs to have a novel component but the novelty also has to represent comprehensible added value for others, not just the originating party. A new definition of the term invention tried to incorporate all the proposed criteria to offer a better understanding of the term in a software engineering context. With a new criterion, expectations of the inventive step would be increased, but the proposed definition could be applied in other industrial fields.

2. Inventions in Software Engineering in Particular

470 In the prevailing literature and judicial practice for patent law, it can be observed that the term of technical invention is interpreted differently in the field of software engineering than in other industrial fields. This is particularly true for the European interpretation.[967] I therefore wanted to know what the software developers considered as innovative in the field of software engineering.

471 It was noted that computer programs were *used similarly to machine tools or analogue processes* as a means to fulfil a specific task or solve a certain problem. The interviewees emphasized that many procedures and solutions currently applied to mechanical and analogue processes could be translated into a digital environment. Computer programs would have the same technical abilities as other technical tools. Differentiating between mechanically and *digitally implemented* technical solutions and teachings, as practised in certain jurisdictions today, does not correspond to the fact that, from a technical perspective, all are applied similarly. In fact, many interviewees explained that software would offer even more technical possibilities than mechanical components, providing inventors with new means to solve problems that could not be conquered before.[968] It therefore opens a new field of technical applications that, due to missing technical opportunities, was closed before. Thus software engineering can generally be used in the same way as non-digital tools to solve a problem, sometimes even providing new opportunities to solve previously unsolvable problems.

472 On the other hand, software engineering as an industrial science is still a rather young discipline in which not all scientific principles or methods are yet

967 See above N 277 ff., for further information.

968 See for example transcript B, N 26, for the argument that it opens a new field of applications.

established. New developments and practices are found on a daily basis as there is room to explore and "to invent the wheel".[969] In addition to the regular spheres in which innovation appears in all established fields of science, in software engineering there is additional room to find new innovative approaches and tools for *how software itself is created or made*.[970] These inventions concern expanding the knowledge of the computer science itself and can, for instance, lie in increasing a program's capability or performance or offering new approaches for a user interface or look-and-feel that increase a program's usability. And as many aspects are still undeveloped, inventions that illustrate and describe technical processes or operations can occur.[971] While a lot of this knowledge development and exchange is observed online, it was noted by the interviewees that few inventions are going to be made in just computer science. They believe it is more probable that "true" undiscovered inventions in products and processes will be found in close development with hardware.[972]

The interviewees provided a long list of discoveries they considered as inventive: 473

- the fast algorithms behind Google Search;[973]
- push messages on cellphones, especially on iPhones;[974]
- a large increase in user experience or usability;[975]
- a new method for graphical representation, especially in the user interface;[976]
- Google Maps for navigation and searching for near locations;[977]
- the use of an app to discover which star constellation or mountain range you are looking at by pointing your cell phone in this direction;[978]

969 Transcript H, N 39.

970 See interesting discussion in transcript B, N 26.

971 Transcript G, N 36.

972 See, for this statement, transcript G, N 36.

973 Transcript D, N 22.

974 Transcript K, N 33.

975 See transcript K, N 35. Another interviewee explained that it is easier to be inventive and try something exceptional when your software has no or very few users; the more that wants of users and third parties have to be met, the more complex the problem becomes and the less the engineer is able to change the existing system and start over (transcript G, N 24).

976 Transcript K, N 35, and transcript D, N 22.

977 Transcript F, N 21.

978 Transcript I, N 62.

- new functions that are offered in libraries to facilitate an engineer's work;[979]
- digitalization of processes, if a new way of presenting an analogue process digitally is discovered;[980]
- a special and new database environment;[981]
- a business model that includes a never-seen selling method;[982]
- discovering an entirely new, undiscovered function;[983]
- new technologies and methods of operating and deployment.[984]

474 Looking at this summary of feedback, we can recognize, again, that the interviewees take into account when something is "new", when it "represents progress from the previous status", when it makes a previously known approach "more efficient" or "more easily applicable" and brings "added value" to a particular audience. What the interviewees considered as innovative in software engineering thus widely correlated with what they considered innovative in other fields of industrial science. The innovative scope in software engineering thus corresponds with that in mechanical or analogue solutions, with the difference that there is still a lot of room for new inventions in computer science.

3. Triviality as Contrary to Innovation

475 One problem closely related to understanding what is inventive is to determine what we consider as 'not inventive'. During the last decade, the reputation of the U.S. patent system has suffered through having an inaccurate patent examination procedure. In many interviews, companies complained that the U.S. had been granting too many *trivial* patents. The term "triviality" therefore has not only become an opponent of invention but it has also had dramatic effects on the recognition of software patenting as a whole. In my research it has been shown that many companies, especially smaller engineering ones, are highly intolerant of granting patents for inventions that do not truly "deserve it". These trivial patents are, in theory, able to restrict the mar-

979 Transcript A, N 58.
980 Transcript L, N 35, transcript B, N 38, and transcript I, N 62.
981 Transcript C, N 26.
982 Transcript B, N 36.
983 Transcript G, N 42.
984 Transcript H, N 45.

ket for software developments as such, giving a monopoly-like right to a good that does not fulfil the legal requirements. I say "in theory" because none of the interviewed companies has ever been successfully attacked by a party holding a trivial patent. Some of the interviewees were wary of other market participants who, in their view, hold weak patents. Even if some of these weak patents would not stand up in court, trivial patents were described as having a remarkable effect on companies which had to take preventive measures or refrain entirely from expanding into the U.S. market. Triviality consequently has an important role when examining the invention term, in order to prevent counterproductive effects on the market.

476 Examining the triviality of an invention is already partly integrated into the European and U.S. patent system when testing under the patent requirements whether an invention is "non-obvious to an expert". Similarly, the interviewees suggested that discoveries should be considered as trivial if an expert would see them as *self-evident*. On this point, the interpretation of non-obviousness and the results from the interviews are coherent. However, contrary to the present legal understanding, an *expert* was not considered an "ordinary skilled person in the art to which the claimed invention belongs",[985] instead, the interviewees defined an expert as an especially skilled person of the particular technology in question. An examiner with any skills in software engineering would therefore not suffice for testing an invention in a particular field of software engineering, if the examiner was not skilled in that particular field, for instance user experience design or a certain programming language. In addition, higher expectations were set regarding the quality of the examiner's skills in that the person examining should have above-average, and not just ordinary skills in the art. This was rooted in the belief of the interviewees that only a deeper understanding of the technology was capable of fully recognising whether an invention truly involved an inventive step (and was not trivial), and would thus be worthy of protection. According to the interviewed parties, the reference point should therefore be *an expert's knowledge with special skills*, and not an ordinary skilled person in the art of software engineering, to determine what was trivial or self-evident.

477 According to the interviewed companies, a higher standard for examining inventions (e.g. in patent law), would, in addition, address another problem brought up by several interviewees. In patent application procedures in particular, it is often difficult to counter vaguely formulated patent descriptions because there is insufficient documentation of this problem in the prior art. If

985 Art. 56 EPC; 35 U.S. Code § 103.

an especially skilled person in the field was examining the invention, their particular state of knowledge and experience could, according to the interviewees, replace missing written evidence of prior art.[986] The examiner could thus reject patents that, in his or her experience and state of knowledge, were not inventive, even if no direct evidence of prior art was available to destroy the claim of inventiveness. Cases where trivial inventions which only summarized prior non-formulated know-how were patented could thereby be intercepted. This would be particularly relevant for certain business methods that would not be considered inventive by an especially skilled expert but today would still have to be patented because of the lack of description in prior art. Expanding the expert term would thus help in distinguishing trivial from true inventions.

478 In addition, the interviewees would want an invention not to be examined solely from an ex-ante position, thinking oneself back into the time before the invention was created. They believed a more realistic evaluation could be made if the skilled examiner tried to solve the same technical problem without looking at the proposed solution. They felt that many developments would be evident if an engineer took enough time to take a closer look at the problem. Just because somebody was the first to try and succeed in solving a particular problem should not mean that this person is rewarded with a patent.[987] In the examination process, the expert should thus follow the approach of thinking back to the time when the patent application was filed and try to solve the problem in question. The perspective thus changes from a merely cognitive approach where it is evaluated whether the invention is non-obvious, to an approach involving addressing the problem and taking the time to think through the problem. An invention would thus change from the formulation "the solution is not non-obvious to the expert" to "if the examining person was given the same task, and tried to solve it, the person would not themself think of the way the inventor solved the problem within a reasonable time-limit". Coincidental discoveries that showed poor quality would be refused protection, if functional factors such as opportunities and technical feasibility were taken into account. Through this approach, an expert could more easily recognize methods or processes that represented logical or simple continuations

986 Transcript G, N 78 ff.

987 See transcript D, N 38.

of prior art and exclude them from protection as trivial.[988] As the quality of protected inventions would increase, the danger of abusive protection-seeking applicants could be lowered.

If both propositions were put together, the full explanation for the non-obvious test would look as follows: An invention is non-obvious, and therefore novel, if an expert with special skills in the particular art of software engineering (in which the invention is claimed) could not themself think of a way to solve the given problem, as the inventor has, within a reasonable time limit. 479

4. Discovering the Innovative Momentum

Currently, patenting focuses on innovations before they are implemented as products that can be commercialized. Contrary to copyright, where outcomes or expressions are protected, patenting aims at an earlier period. In order to discover whether in software engineering this time perspective corresponds with the factual circumstances, the interviewees were asked where on a timeline most of the innovation clusters. 480

The interviewees all said that they observed inventive novelty and enhancement at every step of software development, from new ideas for core features, to the requirement design, to a new technical implementation, up to different testing tools. They consequently were open about when inventions can occur within software development projects. At the same time, they widely agreed that it is in ideation where the innovation actually emerges. This would be where "the inventive peak" occurs and where the "initial impulse" is provided.[989] Everything following is just a consequence of the prior idea, where it is "simply realized". 481

However, the interviewees explained that enhancing and integrating the idea in the concept phase was as important as having the initial idea itself, because this is where it is tested and concretized into a perceptible shape – just as an algorithm represents the technical expression of a particular idea.[990] While the innovative origin is thus situated in ideation, the "purely innovative" phase extends to conceptualization. The innovative momentum is therefore situated in both ideation and conceptualization. 482

988 See transcript E, N 101.

989 Transcript D, N 24.

990 Transcript A, N 58.

V. The Legal Protection of Computer Programs

483 Legal protection of any kind of goods has to be proportionate and reasonable. Not only does the subject of protection need to be chosen with great care but also the various interests behind a statutory solution require consideration. The significance and requirements of a legal system change over time. A framework should be challenged and adapted every once in a while. The following section is concerned with trying to understand which factors behind a legal framework of software protection are important for the software engineers, what expectations and wishes they have in this context, what experiences they have with the current legal system and how they would suggest being protected in the future. It also takes into account the time and event-related aspects by asking the parties how long and to what extent software protection should be offered and embedded in a legal surrounding.

A. The Significance of Software Protection

484 Whether or not software should be protected with IP law is widely disputed. In this section, the interviewees were asked to outline what significance legal protection of computer programs has for their occupation. Additionally, they were asked whether they (or their employer) would continue developing software if no particular legal protection was provided for computer programs. We thus first establish what might happen if no protection for software was provided, and in a second step, which factors influence the relevance of legal software protection overall.

485 Two different groups formed in the interview series: one that felt legal software protection had significant value and supported intellectual property rights, and one that considered legal protective measures as irrelevant for their daily work.[991] The majority of those interviewed, however, stated that legal protection of software was indispensable for their work. Even without legal measures protecting their software, the interviewed parties said they would continue developing and commercializing software. However, they would rely

[991] There were just two interviewees who considered the legal protection of computer programs as a negative thing, saying it represented an unnecessary additional effort they had to make, and today's framework can cause major legal insecurities (transcript A, N 108, and transcript H, N 80 and 92). Asking them what would happen if no legal protection of software at all was available, both clarified that they believed a minimum legal protection for authors and computer programs was crucial but that less would be more (see transcript A, N 112, and transcript H, N 98).

more on alternative security methods, draw up stricter contracts and increasingly enforce these rights, and work predominantly with secrecy in order to protect sensitive information and know-how from outside. They also said they would stop sharing their inventions with society, stop issuing scientific papers on their discoveries and would no longer register any development publicly. They would refrain from going to conventions and would keep know-how exchange with third parties to a necessary minimum. Companies would put more effort into hiding technical ideas and disguising the source code in their product, distributing their computer program only in a closed code form. It was found that although companies could cope with this fictional scenario, all the interviewees showed a lot of discomfort with it. They believed that with this scenario the market would stop sharing its knowledge, technical progress would consequently slow down and that the remaining competition would be 'more relentless and bloodthirsty'.[992] In the long term, this would not bring positive changes. Instead, the interviewed parties believed that it would have dramatic negative effects on the exchange of knowledge within the industry and on the publicity of inventions. It would create an environment that was less constructive, less creative and also less inventive.

According to the interviewees, the main investment in software development is the time, resources and domain know-how a company puts into development and they said they would like the software IP protection to reflect this in some way. The result of the developer team's 'cognitive beads of sweat' should thus be protected from third-party exploitation.[993] Some kind of protection against third parties recreating a similar version of a program or making a simple 'one-to-one' copy of it, would be the minimum prerequisite offered by IP law.[994] 486

Copyright in particular was said to be very important for trading with software. In the absence of a physical element in software, the creators cannot rely on formal ownership to assert proprietary rights to the products. Instead, intellectual property law determines whether and to what extent computer programs are embedded in IP law, and to what degree software is tradeable.[995] Trading is how the software companies make money.[996] The interviewees 487

992 See in particular transcript G, N 92.

993 Transcript D, N 36.

994 Transcript I, N 63.

995 See discussion in transcript G, N 86.

996 Transcript G, N 86.

stressed that IP rights do foster innovation and investment.[997] Apart from protecting a company's software creation, IP law also provides the basic rules to manage the work and determine what one is allowed to do and what is forbidden. The law therefore has the function of setting rough allocations of responsibilities and duties in software development. It represents a guideline for the companies involved.[998] The interviewed parties said they try to respect the potential IP rights of others in their business, and when developing and commercializing their own software.[999] Companies do their best to verify with the available resources and tools whether any third-party IP is affected. They anticipate and undergo a 'cleansing process' in order to prevent potential future legal disputes. The law therefore also has a preventive, defensive character, offering legal security and guaranteeing the assignability of a digital good.

488 Although some companies stated that, at the beginning of the process, legal protection of software would not be of much relevance for them, they said that its relevance would increase when the product was published. During the development phase, protective measures would be of less importance, with the focus on the technical aspects, the project concept and the product financing. Only at a later stage would protecting the product against outside influences and securing revenue become important.[1000] A fear of software piracy from China or Russia in particular was mentioned several times during the interviews and seemed to be quite a hazard for companies that develop and commercialize software.[1001] It is important for the right holders that the law also provides them with simple and effective measures to enforce their rights to intellectual properties. Apart from legal measures, companies at this stage tend to rely on technological security measures to protect their products from being reproduced. All interviewees confirmed that they implement particular technical mechanisms to counter abuse, including the installation of serial codes, concealing the source code with encryptions and working with server-based services where clients do not get access to the actual source code. A certain degree of protection is thus achieved with alternative methods to keep the know-how in-house.

997 Transcript K, N 60.

998 See, for example, discussion in transcript C, N 62.

999 One interviewee described this by saying that his company worked "cleanly" (transcript E, N 175).

1000 Transcript D, N 91.

1001 See particularly transcript K, N 61, and transcript I, N 64.

However, the interviewees explained that there would be certain product or service categories that were less exposed to infringement or imitation. This, firstly, includes products or services that are specially made for one particular customer (individual software), sector or process.[1002] The application in this case is often very specific. If a technical solution is to be imitated on the market, a considerable amount of domain know-how is required, which is difficult to copy from a source code. In this field, legal protection of a program becomes less important. Secondly, with the rise of the service culture fewer computer programs are actually being shared with the customer, instead remaining on the developer's server. As third parties do not get full access to the software's "core", the software is less exposed to infringements.[1003] A similar situation arises in the app market where access is widely controlled by the App Store providers. Organizing and enforcing protective measures in this case is assigned to the provider's sphere of responsibility.[1004] Further, there are many companies that mainly develop for contractually bound customers.[1005] In this case, infringements are limited to overuse or excessive use of the software product, not breaking but overstraining the contractual agreement. This kind of conflict was described as happening on a regular basis. But as the developer can react on the basis of an existing customer relationship, the problem can usually be resolved rather quickly. In most cases, the culpable party was said not to have known that they were overstraining the contractual terms. In a contractual relationship, suing infringements and stopping third parties from copying, thus, is of minor importance. All these scenarios show that the IP rights' function to minimize infringements is mostly limited to third party-abuse. Contractual breaches are often resolved in another way. Apart from that, the interviewed companies valued the IP rights' function to define software as a tradeable commodity and provide delimiting rules. It seems that in the case of computer programs the main function of IP law is to create and assign property rights instead of actually enforcing them. 489

All in all, the interviewed parties agreed that legal protection of software was desirable and necessary to protect the developer's or right holder's interests. Without this protection, the interviewed companies would make greater use of alternative sheltering mechanisms, such as secrecy, concealing technological measures and contractual barriers. In this case, know-how exchange would 490

1002 See transcript I, N 6 & 16.

1003 See interesting discussion in transcript A, N 114.

1004 Transcript K, N 61.

1005 Transcript I, N 28, and transcript G, N 65.

be strongly limited. It was discovered that IP law serves several functions and that in particular the assigning and regulating function of IP is important for right holders. Ensuring merchantability is more the focus than enforcing rights and fighting infringements. The protective functions of IP law become more important after a work is published on the market and becomes accessible. The exposure of software is limited where a product or service can be considered highly specialized or is less accessible to the public. In this case, legal protection becomes less important.

B. Experiences with the Current Legal Framework

491 The findings of the previous sections revealed that legal software protection of some kind continues to be wanted and supported by the software developing companies. This section seeks to determine, from the perspective of software engineers, what is relevant in a legal framework (necessities), and what could be improved (wishes). This was established by asking the interviewees about their past positive and negative experiences with the current legal framework with regard to software development and protection.

1. Positive Aspects

492 The interviewees confirmed that, by and large, they were satisfied with the current legal situation. The results of international endeavours were perceived in the industry.[1006] Although several suggestions for improvements were offered and some regrets were expressed, companies appreciated that software was enclosed in the protective mantle of IP law. They appreciated that they were able to develop their products in a free market and trade them without formal barriers. The interviewed companies considered the current law to be quite reasonable and balanced. Legal entities appreciate that they are not greatly restricted or limited by the regulatory system and that the law in Europe and the United States generally provides them with the tools they need to commercialize and protect their works.[1007]

493 The hybrid model in computer program protection, embedding both patent law and copyright was considered to be widely acceptable. Copyright especially, with its copy guard and its reference to the author was highly appreciated.[1008] The applicability of copyright and its requirements to computer pro-

1006 See particularly transcript M, N 35.

1007 Transcript D, N 46.

1008 See particularly transcript I, N 110, and transcript G, N 98.

grams therefore was undisputed. More criticism was directed at the patent law system. Although protecting inventions is right and necessary, patents could in some situations present a high barrier. Nevertheless, patent law in general was considered to be reasonable and understandable in principle.[1009] The interviewed companies said that in particular the stricter European patent system gives the right incentives (although partly with the wrong methods), while the U.S. system requires some fine tuning. On the whole, though, the companies believed that both the copyright and patenting of software were accepted and established throughout the industry.

2. Negative Experiences and Suggestions for Improvement

Although the basic conditions were considered suitable, the interviewees mentioned several aspects that did not meet their practical needs. The next section highlights these points with the potential for improvement to meet the expectations and wishes of modern software development and commercialization. 494

The interviewees discussed several aspects within this problem field. For example they noted that the existing procedural laws were unattractive to defend their rights against infringement.[1010] Several described how the current disclosure requirements in patent applications were insufficient and that not enough relevant know-how was shared with others even though the invention was granted an exclusionary right afterwards.[1011] One interviewee emphasized that the current definition of the term “interface” in copyright was outdated and caused uncertainty about the rights and obligations of the provider.[1012] Several interviewees stated that the rights of the users should be strengthened once a product has been purchased, because today customers are constricted by extensive licence agreements, forbidding the users from conducting minor changes necessary to maintain the computer program.[1013] Stricter exhaustion rights and a stronger restriction of the right holder’s power would therefore be desirable to empower the user. 495

1009 See discussion of above in N 415 f.

1010 See for example transcript B, N 88.

1011 See discussion in transcript A, N 126.

1012 Transcript C, N 70.

1013 This point was completely new but very interesting from a user perspective, found in transcript H, N 106.

496 Among the points brought up by the interviewees, the following four aspects were particularly of interest:

a) Computer Programs as Intellectual Properties

497 Many interviewees described how they believed that *software* as an intellectual property *has not yet been fully understood and accepted in law*. The fact that software is built up on digital operations distinguishes it from all other intellectual property goods, which can be easily expressed in a physical, touchable medium. Software can be run in a physical medium, too, but the technical mechanisms and processes involved are not necessarily observable in a physical form. Apart from the hardware-related computer programs, the only expression one may get for software is a "blank page with scripted commands". This lack of a tangible physical representation was said to make software as a medium particularly hard to depict and analyze in law.

b) The Time Factor

498 The fact that software is a digital product makes it particularly easy to reproduce and distribute it. The consequent higher economies of scales were described as being enormously helpful to get established in the market faster and to regain the investments within a shorter time. The market participant's goal is to reduce the time-to-market during development in order to reach and settle in the market first and faster. As lead time becomes the major factor in commercializing software, providing a legal system with better-structured and less time-consuming administrative processes could support software developers and right holders in meeting their timelines and getting into the market quickly. During the interview series, several parties described how the administrative processes connected to granting exclusionary rights to computer programs currently took too long. In the field of software, the pace of product launching is said to have picked up substantially over the last ten years. The time lags due to administrative processes thus have particularly negative effects, as they consume a large quantity of resources for no particular reason or only little gain. A legal framework for software that takes the significance and value of time into account would be desirable.

499 Time is not only a determining factor when rights are granted but also afterwards when third parties are excluded from using and imitating the product. All involved companies agreed that copyright's current term of protection of 50 years after the last participating author's death would be far too long for a

dynamic and short-lived product such as software. After 50 years, the protected original would not be instructive or useful for the public domain in any way. The know-how exchanging function of IP would thus be forfeited. In some of the interviews, we tried to deduce whether to date there has been a computer program in the United States or Europe that has actually reached its end of protection and whether a product exists that in fact would still work properly after 50 years of utilization. Depending on the definition of a computer, it could be said that the first, and probably the only 'computer program' that has ever reached its protection limit was Alan Turing's machine 'Christopher', created during the Second World War to crack codes of the Nazis. As Alan Turing died rather young, in 1954, the program 'Christopher' came into the public domain at the end of 2004, though one may dispute whether Turing's early program really falls under the definition of a computer program. Computer programs with higher programming languages and source codes only emerged at the end of the 1950s. According to the interviewees, probably only a few of these programs would still be useable on a modern computer. From a technical point of view, a term of protection of 50 years would seem too excessive. A shorter term would be more realistic and would better meet the society's need to participate in a protected creation or invention. Several interviewees remarked that they did not understand why a computer program's term of protection in most statutes was tied to the author *in persona*. They argued that most computer programs are developed by companies and entities. In this scenario, the developer commonly has to assign all IP rights they may hold on a development to their employer or principal. With some employment law, this happens automatically.[1014] Tying the term of protection to the author's life span rather than to the moment of creation or to the developing company thus fails to match their needs and the actual circumstances. According to the interviewees, this problem gets even more complicated if we consider that in most companies a large number of developers contribute to a product. Theoretically, a company would need to attribute each code line to an author and record for each software product when the last co-author died, in order to calculate the term of protection for an ever-changing product such as software. Tying the term to the author thus causes various practical difficulties.

[1014] According to Art. 17 Swiss CopA and § 69b Abs. 1 German UrhG, the employee automatically cedes his or her author's rights to the employer, if he or she produces a computer program in the context of his/her employment relationship.

c) *Commissioned Work*

500 Whenever a company offers a contract for a project to several companies, the developers are quoting and competing against their direct competitors. The companies here often put a lot of work into their offers. The more detailed the project is in its tendering, the more elaborated the concepts and drafts of the bidding companies usually are. The mandating company can then choose between several highly elaborate offers and select whom they want to commission.

501 Several developers who regularly place bids for commissioned work have experienced that potential clients let them present their elaborate ideas and concepts, and then afterwards utilize and imitate the presented concepts without actually commissioning the developers.[1015] Some have even experienced this behaviour with governmental institutions.[1016] Principals would often choose the cheapest offer but still use the inspiration they got from other biddings. Companies bidding for commissioned work seem to be exposed to the danger that the ideas they present may get abused at the pre- or non-contractual stage. This could in theory be dealt with by signing a non-disclosure agreement or contractually agreeing on a prohibition of use before pitching. However, in practice, most principals do not want to sign a non-disclosure agreement. Instead, they want to get full and (legally) unrestricted access to a technology so they can analyse, review and test it.[1017] Some principals are subject to particularly strict regulation, such as security or data protection law. They then have to conduct tests to verify that the products meet the requirements. Ultimately, the principals have the power to decide who receives an order and thus the turnover. For this reason alone, no developer wants to bother them with contractual prohibitions.

502 As the bidder who does not get the mandate rarely holds any enforceable rights on their concepts and can hardly monitor what happens with the presented drafts, the presented concept is often dissipated. The interview parties felt that the copyright framework today does not adequately guarantee the protection of concepts and drafts. It is in this preliminary phase that enhanced

1015 Some software companies noted that they would watermark all documents handed out to principals with the words "rights reserved" or "copyrighted by...". However, during the interviews they could not say with sufficient certainty to what extent copyright in the concepts actually existed and whether these would have been legally enforceable.

1016 See transcript B, N 86.

1017 Transcript D, N 48.

concepts and drafts are exchanged without a binding and protecting contract. Project developers consequently see themselves in a very weak legal position. They cannot prevail against potential future clients, even if those clients do them wrong.[1018] The companies therefore wanted better protection of their work results in intellectual property law in the preliminary phase of a contract or final product.

d) Patentability of Computer Programs

The eligibility of software patents is where my research originally started. This topic is politically disputed in Europe and the United States. No proper solution has yet been found. While Europe is discussing (officially) opening up patents for computer-implemented programs, the United States is looking for a way to restrict them after negative experiences with patent trolls and trivial patents. Although both patent systems have a similar basis with related protection requirements, their practical implementation appears to be completely different.[1019] 503

In the interviews, the discussion as to whether software should be patentable was unexpectedly emotional and partly based on contradictory arguments. As described above,[1020] during the first interviews of the research series, many small companies described patenting of computer programs as a bad and preventive matter. However, it was only after the implementation of an additional question into my interview guideline that it was discovered that, in fact, none of the representatives of the eleven different companies had ever experienced or heard of their company really being hindered by patents in developing or commercializing their computer programs. No interviewee stated that, if the product represented a true and qualified innovation, patenting computer programs would be unreasonable or damaging or preventive for their company or their work.[1021] According to the interviewees, an invention would deserve to be protected if it was truly new and originative. Instead, their criticism was focused on the possibility of patenting trivial innovations and on the rights and obligations of an exclusive right holder. They closely linked the question of software patents to the effort they had to make to ensure that they did not in- 504

1018 See transcript D, N 34.

1019 For the interpretation gap, see transcript F, N 95 in particular.

1020 See N 415 f.

1021 See for example the turn in transcript I, N 110.

fringe third-party rights during the development and commercialization of their own products. These preventive measures were experienced as time-consuming and negative.[1022]

505 What can be said from this observation is that the problem is not about patenting software in general. This question was undisputed among the interviewees and they agreed that software should not be excluded from the patent scope. Instead, it appears that the range of the scope itself (innovation vs. triviality) and the bundle of assigned rights (rights and obligations of a right holder and a user) were considered controversial. This insight is important because current discussions in Europe and the United States focus on a system question instead of discussing the framework of patenting in general.

506 On the subject of software patenting, the companies highlighted four main problems they had experienced as negative in the current legal system:

aa) Machines vs. Computers

507 One major difficulty in evaluating software patents is the inexpedient definition of the term 'technicity' under the European Patent Convention. Even for lawyers it is not beyond doubt to what extent software can be patented in Europe. The current political debate mainly distinguishes between mechanical (analogue) and computer-implemented (digital) innovations.

508 The software engineers and developing companies agreed unanimously that computer programs, even if not connected to hardware components, fulfilled the technicity requirement themselves: "In reality, every software product or real world conception of software (...) goes way beyond the mere algorithms and has a technical effect or conjunction (...)."[1023] Every practice asking for implementation or connection to a hardware component was described as unnecessary and restricting.[1024] Developers believe that the current difficulties of law in defining the technicity term are partly caused by a lack of trust: "Personally, I would expect a better acceptance of the information technology industry".[1025] Software developers believe that society does not truly understand

1022 In this respect, the parties referred in particular to searches by patent attorneys and other experts on the basis of patent applications, registers and similar sources; see discussion in transcript B, N 90.

1023 Transcript M, N 39.

1024 Companies described the "trick of connecting the program to a computer program" as akward but helpful in order to "bypass the law"; see for example transcript C, N 82.

1025 Transcript G, N 104.

the subject and thus is mistrustful. From the perspective of the interviewees, software was currently the most innovative field, not just for computer problems but rather for all technical problems. They emphasized that it would be important for all technical industries for software to be integrated into a modern patent system, globally and in a uniform manner:[1026] "To close our eyes to this development would not be expedient".

About half of the companies stated that computer programs should be granted full protection under the European Patent Convention. Even the others who were critical about patenting in general confirmed that there was no technical reason to distinguish between analogue and digital inventions. Several parties, including specialized attorneys and patent attorneys, were uncertain whether and to what extent computer programs were currently patentable in Europe. The European statute seemed to cause great uncertainty and was said to interpret technical requirements in a way that was not comprehensible for experts in the field. 509

bb) Requirements for Patenting

Overall, the interviewees comprehended and supported the statutory requirements for patenting.[1027,1028] They recognized the difficulties a legislator faces in defining what should be considered as patentable in software engineering. Some observed that it is difficult for patent authorities and lawyers to evaluate the inventive step of a computer program because the novelty and non-obviousness are hard to verify and test.[1029] Consequently, the statutory requirements for patenting, although uncontested in theory, are difficult to apply in practice. Some interviewees, especially the representatives of smaller or medium-sized companies, expressed difficulties in understanding the necessary terms. They described the requirements to be "difficult to measure"[1030] and "too vague"[1031]. Clarification in the form of clear definitions and a catalogue with examples would be helpful in this respect. 510

1026 See for example transcript I, N 102.

1027 For more information regarding the legal requirements for patenting, see above, N 310 ff.

1028 The only exception to this finding is in transcript H, N 110 ff., but I assume the candidate misunderstood the question.

1029 See for example transcript G, N 108 ff.

1030 Transcript K, N 67.

1031 See transcript E, N 137, transcript A, N 130, transcript F, N 67, transcript L, N 80, and transcript B, N 97.

511 The interviewees said they did not fully understand why the present patent requirements were not tested substantively in each patent application of every country. To increase legal certainty for both the right holders and third parties, the interviewed parties suggested that an in-depth substantive examination and testing of the patent requirements should be conducted as soon as a patent enters the application procedure. The main reason for this is that the innovative patent holders, too, believe that patenting trivial innovations should be avoided.[1032] By testing each patent requirement separately – prior art and non-obviousness in particular – patenting trivial innovations would at least be minimized.[1033] Although closer testing of the patenting criteria may require more time for the application process, the companies believed that the additional legal security would be worth it. If IP law thus continues with the existing patent requirements, the interviewees believed these requirements should at least be verified substantively before the right holder obtains an extensive patent right.

cc) *The Power to Block and Abuse*

512 Many of the interviewees emphasized that *software patents might have a greater effect on the engineering market than other types of patents*. This is rooted in the fact that good software solutions prevail quickly, and as everybody wants to build their program with the newest technical solutions, a patent blocking this method or technology could prevent others from keeping up. The most efficient solution to a technical problem would consequently be obstructed. Likewise, trivial developments based on best practice but not truly inventive might have a similar effect as third parties are blocked from using the same approach, although they are technically required to use it. In both cases, the right holder obtains a powerful exclusionary right while others are prevented from using the innovation. It is the duty of a legal framework to address these problems.

513 In the media, we often hear about so-called *patent trolls* that 'abuse' their exclusive rights in court to demand high compensation from third parties. Several interviewees explained that companies had contacted them claiming that their patent had been infringed. All those companies that had not been confronted with such assertions were afraid that the same could happen to them some day. Although some suggested that these claims probably would not

1032 See above N 475 ff.

1033 See the same idea in transcript D, N 99 ff.

stand up in court, they believed that defending their company against such accusations would require a lot of financial effort and time resources.[1034] Smaller companies especially are unable to provide these resources, even though they may be in the right. Patent trolling, not only in the field of software, but in the whole patent system was identified as a huge problem and a challenge of our time. While the problem of patent trolling was described as having escalated in the United States, Europe and its restrictive patent system was said to have it better under control. One of the interviewed companies said that patent trolling was encouraged by certain civil procedure codes; deficiencies in the litigation system were said to foster the enforcement of weak or illegitimate patents even against admissible activities.[1035] They said that this problem could partly be solved if the plaintiffs were required to contribute to the lawyer's and process costs in case of defeat.[1036] In this way, the system could at least partially get a grip on the effects of unnecessary litigation.

514 The industry takes its own measures to react to the issue. It implements preventive measures in the development process, which guarantee a clean production line in which no infringements should occur. It also raises the level of documentation in order to be prepared in case of a legal dispute. But they state that all these preventive measures would still not protect them from being confronted with unjustified and resource-consuming claims. They clearly expressed a wish for further effective legal defence.

dd) Patenting as the Discipline of Kings

515 During one of the first interviews, one representative of a smaller company complained that today's IP protection of computer programs was tailored for big companies who can afford to mandate an attorney.[1037] The interviewed companies said that, nowadays, only bigger companies were able to supply the necessary means to protect and enforce their goods with and against patents. Patenting has, thus, become a "discipline of kings",[1038] locking out smaller businesses. According to the interviewees, the U.S. market is particularly risky and unattractive for smaller developers. On the whole, patenting was described as being of very little value for smaller companies, as the protective effects of a

1034 See for example transcript D, N 30.

1035 Transcript M, N 35.

1036 See the description of the upcoming Innovation Act in the United States in transcript M, N 78.

1037 Transcript H, N 122.

1038 Transcript F, N 51; transcript B, N 145.

patent was small compared with the expense required to get one.[1039] The resources of smaller companies are limited in terms of labour, time and money. Each case of illegitimate use has to be evaluated carefully. The first priority of a start-up is to get customers and start commercialization.[1040] Rights management and enforcement is not their main priority. If a new project was launched, smaller companies explained that they would be able to do very little research, ask few questions of legal counsels and only spend a little money on preventive measures. Instead of mandating a patent attorney, smaller legal entities usually do their own searches with very limited possibilities. Their results would often not be representative, and would thus cause legal insecurity for these companies.[1041] Smaller companies are often not able to take preventive measures during the development of a computer program. This fact makes these companies more exposed in case of litigation.

C. The Optimal Framework

516 Knowing more about which things work well in the current legal system and which are improvable, we can try to deduce a new optimal framework for computer program protection.

517 The interviewees were instructed to try and forget everything they knew about computer program protection in the current legal framework. Ignoring what is implemented in today's copyright and patent law and where the difficulties of both systems lie, they were asked to picture an optimal and balanced IP system for software.

518 In a first step, the candidates tried to determine the function of a legal framework in protecting computer programs and to isolate which characteristics had to be served. Also knowing which components of software were most valuable for software companies, the next step was to determine which parts of software they believed should be legally protected. Finally, they identified the starting point for the legal protection of these elements and, from a time and scope perspective, where protection should end.

1039 Transcript B, N 40; transcript K, N 100.

1040 For more information, see transcript B, N 40.

1041 See, regarding this problem, transcript K, N 100 and transcript I, N 130.

1. The Function of Software Protection

519 Law should intervene in the market only with good cause. For example, a protected subject or object should only be legally protected by protective measures if there is a need for it. The following section explores which functions a legal framework for software protection should serve from the perspective of the software companies. For this purpose, the interviewees were asked why legal protection of software was important to them as software developers or companies commercializing computer programs. They emphasized the following key functions of software protection.

520 One major criterion for the interviewed software companies was that *investments* that were spent on research and development should be protected in some way. Whenever a company has to allocate money, it needs to ensure that it can protect the final result and retrieve the sum spent and a bit more for future investment. Legal investment protection consequently implies an ex post perspective, focusing on the time when a company tries to commercialize its product or service. But it also has a perspective on future events, as potential gains are used to keep the company running. The interviewees emphasized that the economic interests behind financing IT developments would differ from ones in other industries, where IP law was better established. All the more important would be the protection of investments. Human and financial resources are very significant in engineering. Companies look for the possibility to protect their developments from the start, so they find investors that can offer the money required and the developers can take a chance in the market. The risks are enormous, not knowing whether the market will accept their products and whether they are going to be successful. The companies said they would like a legal mechanism that protects them during the time before the consumer market has decided whether or not a product is interesting enough to buy, use and/or copy. The method of protection should ensure that the right holder can make money with a successful product that compensates for the possible previous or following unsuccessful attempts. Offering legal protection for investments in creative and inventive developments was described as being of major importance for all the companies.

521 Legal protection can also work as an *incentive*. It can "empower companies to continue being innovative".[1042] The creating companies get a reason to invent and stay creative, as their results can be protected by legal measures, and

[1042] Transcript C, N 62.

hence do not have to be disguised in order to protect them.[1043] Software developers receive an incentive to elaborate and share their contributions. Third parties profiting abusively from their work, so-called free-riders, are stopped.[1044] Legal protection also helps to diminish potential risks and fears that may discourage a company from investing in their development.[1045] The interest in continuing to develop is thus perpetuated. At the same time, the creations and inventions are shared with society and industry. Third parties can observe, learn and use prior knowledge.[1046] By offering the creators an option to obtain a return and a way to be protected, they are encouraged to share what they have discovered, and have no reason to hide their findings from others.[1047]

522 Law should somehow display the *work's connection to its author* or inventor because it carries the signature of its creator.[1048] According to most of the interviewees, this perspective is reflected in today's moral rights in copyright and inventors' rights.

523 The final aspect of computer protection is to find an adequate *balance* between the different interests. An optimal framework should reduce the transaction costs of the involved parties. It should create rights for the originator and make it possible to enforce those rights with an appropriate measure through litigation.[1049] The creator wants to have a minimum level of power and control over his or her work. They want to make important decisions such as when and where a work will be published for the first time. But the interviewees also acknowledge and understand that there have to be limits to the absolute rights of a right holder. The users want an adequate benefit for the money they have paid which is usually to obtain a protected right to use. The question is, where can the boundary be set between 'sufficient' and 'excessive' rights of a right holder and which rights are really necessary for a customer. Interviewees described how they were forced to warrant future orders, how they had to accept unsatisfactory products because they were forbidden from

1043 Transcript C, N 62; transcript M, N 27.

1044 See discussion in transcript F, N 81.

1045 See discussion in transcript I, N 115.

1046 See transcript H, N 104.

1047 Transcript F, N 77. If this would no longer be guaranteed, companies might discover other revenue models as described with services. However, these models do not include sharing knowledge with society (found in transcript F, N 79); see also discussion above in N 485.

1048 Discussed in transcript A, N 112, and transcript H, N 116.

1049 Transcript M, N 41.

making modifications and that they even had to accept when the licensor reduced functions and services that were previously provided.[1050] The challenge of the law would be to find an adequate balance.

The interviewees described how the functions to protect a good by law should be dynamic. They should be subject to political, economic and social change and should be reviewed and adapted on a regular basis. It has to be verified whether the law still provides the necessary means to meet these changes.[1051] The software market was described as exhibiting a particularly high pace. Dynamic mechanisms to apply or improve the law would thus be desirable. The product software is no longer a fixed good. In contrast to books in copyright or drugs in patent law, computer programs do not remain unchanged once they are published. They were described as representing *moving ground*.[1052] Talking about served functions in software should thus also involve determining at which point these functions should be served and whether or not the protected good has to have a fixed form at this moment or not. Which functions law serves consequently is a very complex problem. 524

2. Potential Objects of Protection

One of the main goals of my research was to discover what in software development should be protected with intellectual property law. As I wanted to open the field to all possible outcomes, I did not start with the elements known from copyright or patent law but instead tried to discover any potential object that could be sheltered. To find possible solutions, I asked the interviewees what they considered as particularly valuable in a computer program. I then tried to determine which elements in a computer program should be protected legally according to the interviewed companies. 525

a) Valuable Components

I started by asking the companies what they considered to be the 'crown jewels' of the software they develop and commercialize, what has to be protected safely from third parties, and what would cause most harm if destroyed or stolen by others. I asked the interviewees to focus on the elements that would 526

1050 See for example transcript H, N 96, 106, 118 and 132, and transcript E, N 142.

1051 See for example transcript A, N 112.

1052 Transcript K, N 77.

hurt the most if attacked, irrespective of whether this was a financial, personal or very different type of hurt. Either way, the aim was to find out what companies regarded as valuable in software development and commercialization.

527 Their feedback varied and included business plans, source and object code, database designs, user interfaces, the final product, algorithms, the number of functions and features integrated, innovative twists, personal data, product specifications, the look-and-feel, and the final implementation of vertical domain know-how in a software product. They described how the value of an element or component would largely depend on "the business and particular piece of software".[1053] They explained that the valuable components in software could not be generalized for every type and piece of software. Instead, an evaluation of every good would be necessary. Only with this approach a solution that takes into account the particularities of a creation or invention would be achieved. They further stated that it was complex problems that were sensitive to infringements.[1054]

528 Based on the feedback I received, I was able to group certain software components that appeared to be particularly valuable. The elements that were named most were the source code, the algorithm, the integrated functions of features, the look-and-feel and, finally, the integrated vertical domain know-how. They explained this as follows:

529 The *source code* is what makes a program run on a computer. It represents the description of a command that the machine has to execute. It therefore is the technical backbone, which displays a process digitally. Everything is contained in the source code: a program's functionality, the algorithms, the processes, etc. The organization of the code itself in a structure requires a lot of resources and brainwork to build. When a computer program is published, a large part of the technical and practical knowledge is made public and, thus, becomes accessible if it displays the source code openly. In this case, it cannot only be read but also copied. This makes the source code vulnerable to abuse.

530 The *algorithm* was described as representing another key part in software as it mathematically portrays the problem that the machine has to fulfil, its behaviour. It tells a machine how, and on the basis of which parameters, a process should be steered. As it has to be a general mathematical rule for many

1053 See transcript M, N 25.

1054 Transcript I, N 80.

possible situations, it is usually highly elaborated and optimized, requiring a lot of time resources to construct or adjust it. A qualitatively high algorithm requires and contains a lot of vertical domain know-how.

The *look-and-feel* is the way the software is perceived by the user. It contains all the conceptualized small twists that a user may not explicitly detect but which help to guide users and increase the user experience. We therefore talk of a combination of particular aspects of an animated and interacting user interface and certain functions and features that a user experiences. The look-and-feel includes a lot of creativity as it is influenced by the designer's taste, experience and know-how. It is very important from a user perspective and has a big influence on their purchasing behaviour. 531

The *number and combination of various functions and features* determine what purpose a product serves and how well it meets the requirements and needs of the users. It therefore has an important role in shaping the final product. The functions and features are usually sketched in conceptualization, marking the cornerstone of the software, such as velocity, reliability, the front end, data security, the number of integrated business processes etc. These are some of the main characteristics that are compared with the products of a competitor and therefore influence whether a customer buys the product or not. If important functions stop working, the whole software fails, similar to a gearwheel.[1055] At the same time, the originator will have difficulties in commercializing their software if the software's functions and features are imitated in a competitor's product. 532

However, the most named aspect was the *vertical domain know-how* that is integrated in software. Every company has its own special know-how, how to realize projects best, which factors interact, how to detect user needs and wishes, how to offer a one-in-all solution, how to use the system environment effectively, and so on. This particular type of knowledge is used in the development process and has a major effect on the final product. 533

The five most named valuable components in software engineering were therefore the source code, the algorithm, the look-and-feel that is revealed to the users, the combination of particular functions and features, and, finally, the vertical domain know-how that is integrated in a finished program. All five software components represent the company's creative or inventive contribution and are, thus, highly valuable for the developing companies. According to the software companies, these goods represent the particular know-how or 534

[1055] Transcript F, N 37.

knowledge of the company which is apparent in the final program. This makes these components vulnerable to infringement, if no further preventive measures are taken. Some components are not based on knowledge alone, but utilize a trial-and-error approach. Vital for discovering the best solution possible is therefore the time spent on programming, having ideas, realizing and testing them, failing, trying again, and succeeding.[1056] What makes these components valuable, therefore, is the amount of resources put into them. All the attempts needed to get to this point wouldn't be recognizable in the final version. However, competitors could copy the successful version and be spared all the effort the originator put in to develop an optimal solution. The particular know-how and spent resources are therefore both important factors in estimating the value of a software component. It was noted how sensitive both aspects are to infringement and that they have to be kept safe somehow.

b) The Suggested Objects for Software Protection

535 The section above dealt with the question of what the valuable components are in software engineering. The next step is to discover which components would be suitable for legal protection. I therefore asked the interviewees which components in a computer program were worth being legally protected.

536 The replies corresponded with the above results in that invested resources and know-how were not only valuable but also should be protected with legal measures. The collected data suggested that the source code, the algorithm and the look-and-feel contained a particular amount of valuable and sensitive data.

537 All the interviewed parties agreed that the *one-to-one* or *slavish* copying of the whole software product via data transfer or replicating a carrier medium represented economic and ethical damage and should be forbidden. Slavish copying would have the same effect as plagiarism in literature.[1057] According to the interviewees this is particularly aggressive and bold behaviour, not adding any creativity to the final outcome.

538 The companies described the *source code* as containing a lot of technical knowledge, expertise and domain know-how. Further, as the source code contains the technical instructions to the machine, a great amount of time, intellectual and financial resources must be invested in its development to achieve

[1056] See discussion in transcript D, N 36.

[1057] Transcript C, N 38.

a practicable final version that is able to fulfil the desired tasks. The source code is the product of a long process where both knowledge and resources have been invested. In an open code version, all of it is visible on the surface, making it particularly vulnerable. By copying the source code somebody else has elaborated, the program's know-how and utilized time resources are abused, diminishing the value of the original.

Algorithms were described as containing the most and purest amount of know-how. They are the embodiment of the domain know-how in a process that is represented in the form of an algorithmic function. They are built on the developer's experience in successful formulation. This high-concentrated know-how should be protected. 539

The *look-and-feel* was described as the embodiment of the user experience itself. It includes the final version of the software's *functions and features* that make it distinctive and characteristic to use for the customers. But it also represents the visual appearance of a software company, its conceptualized structural perception, its desired feeling. The look-and-feel is therefore closely connected to a company's corporate identity.[1058] Companies want to prevent third parties from using their look-and-feel for two reasons; first, the implemented know-how, which is visible on the surface's look and behavioural feel needs to be protected. But the companies also want to prevent others from abusing their corporate identity and creating a connection to third parties that does not exist. The interviewees in this context talked about user traffic or attention that is skimmed by competitors, deceiving potential clients.[1059] This puts users at risk as they do not receive the service, including the security guarantees, the original company would provide. As software copyright protection today does not explicitly cover this problem field, companies instead rely on branding, design rights and unfair competition law to protect the look-and-feel of their programs. 540

Software components usually come with additional *records and development documentation* such as instructions, commentaries, descriptions and so on. According to the interviewees, these records contain a lot of creative thoughts and know-how. In day-to-day business this type of information is only shared 541

[1058] Transcript F, N 69.

[1059] See for example transcript A, N 70, and transcript K, N 71.

if protected under the shield of a contractual agreement. But in order to protect them from attack by an unrelated third party these documents would require additional IP protection.[1060]

542 The companies stressed that the model of protection should be related to the achievements of the particular software piece as well as to the work and resources that were engaged to develop it.[1061] The achievement relates to the added value or inventive contribution a component has to offer. Software engineering involves a lot of human effort, time and financial resources. As a lot of the involved know-how is visible in the final products, advanced protection is particularly important.[1062] Otherwise, the copier would not only profit from the displayed know-how but also from the effort and resources that the original developers had to invest. Therefore, the legal cover should not only protect the visible know-how but also the investment involved in the development of the software. The know-how and other sensitive information can be grouped into *technology protection*, and the financial and time resources spent, including how creative they were, into another group of *effort or outcome protection*.[1063] Depending on the purpose a legal system intends to serve, either or both groups could be tackled.

543 One of the interviewees further suggested that an explicit minimal effort in developing has to be outlined so that the good is granted legal protection. The effort could either be monetary or time taken to develop the product. But exploring this suggestion further, it was concluded that this would not take into account various borderline cases and would, thus, not be so helpful for defining the scope of protection.[1064]

544 To conclude, it was suggested that it should be evaluated for each case which elements were valuable and sensitive enough to be protected. Several components, including the source code, the algorithm, and the look-and-feel (including both the user interface and certain functions and features) were considered to be worthy of legal protection. All components exhibited a high concentration of know-how and demanded a lot of (human, financial and time)

1060 See for example transcript C, N 40.

1061 They spoke of the relationship between the subject and its protection and the reasonableness of it. See for example transcript B, N 74 and 78.

1062 For more information regarding the visibility of know-how in software components, see above N 444.

1063 Elaborated partially with the help of transcript L, N 94, and transcript G, N 133.

1064 See transcript B, N 74 ff. This criterion was touched on in several interviews, but was not supported in any as a final criterion.

resources for development. These components were identified as the most valuable ones for companies. For this reason, the law should provide adequate protection and efficient measures against their abuse. Depending on the purpose of the legal protection, a particular model for protecting the involved technology or a model to protect the invested effort could be implemented. Additional elements, such as development documentation, drafts and instructions, could be integrated in a future system. Thus the law would recognize software as a product with many different elements and would therefore consider all its different parts,[1065] which would help to provide a coherent and functioning software protection system.

3. Starting Point for Legal Protection

The starting point for legal protection is the moment when an intellectual good becomes recognizable from a legal perspective: People can derive rights from it, build obligations on it and trade with it. In the current legal system, both copyright and patent law focus on protecting the final product, such as the source code in copyright or the elaborated algorithm in patent law, when the development process has mostly been completed, and the components are fixed in their definitive shape and implemented in a final product. During the interview series, the question was asked at what point legal protection should start and what the minimum requirements for legal protection might be. 545

During his interview, one person stated that if he was able to make a change in the law he would want a clearer starting point in legal protection of his software products and services. He stated that he did not know for certain at which phase of the project process legal protection began: "You keep on developing, and at some point protection is simply granted."[1066] When discussing this assertion with other interviewees, many of them knew from their work that the 'expression' currently represents the relevant starting point for legal protection in copyright. However, the interviewees could not agree on the term's significance. They tried to deduce where in the development timeline one could set the point that a work became an expression, but kept switching back and forth between a few sketches during conceptualization, to more detailed drafts or the final source code. It gave the impression that they were trying to refer to a term they had heard before, without truly knowing what it implied. 546

1065 Discussed in transcript K, N 106.

1066 Transcript C, N 72 ff.

547 When the company representatives were asked what they, broadly, considered as an optimal and adequate starting point for legal protection, most of them indicated that, in theory, it would be the idea that included the most creative or inventive activities. At the same time, they expressed some discomfort with granting legal protection to ideas, as visions were very abstract things.[1067] The interviewees thus believed that a development project should exhibit a minimum level of maturity and meaning, and that the good should have reached a minimal stage of consistence to be granted legal protection.[1068] The interviewed parties thought that the idea should be specific and comprehensible enough to outline how it should be technically applied or integrated in a specific surrounding, so that society can also learn something from it.[1069] In ideation alone, the creation does not provide enough information to society to actually profit from its disclosure. Instead, the interviewees stated that the earliest point for protection should be when the idea is 'minimally fixed' into a specific draft of possible use, because with a draft the idea becomes more specific and, thus, loses most of its abstract character. The longer the wait along the time axis, the more the software becomes elaborate and takes a particular shape, making it seizable.

548 The question is how far down the development to set the starting point for legal protection. If it is set too early, it can open the door to abstract ideas rather than realizable solutions. If it is set too late, creative and inventive concepts that are shared with the public remain legally unprotected, or are not disclosed at all.[1070]

549 The software companies felt that software protection should be granted either from the conceptualization or realization stage, with the majority in favour of the conceptualization stage. More than half of the interviewees believed that in this phase most creativity was materialized. Many important decisions that are essential for the final software project are said to happen during this period:[1071] The company produces first sketches and builds prototypes. Through this, an idea is sharpened, receives its decisive shape and gets thoroughly tested.[1072] At the same time, the interviewees said that it is during conceptualization that most brain and planning effort as well as most of the human, time

1067 Transcript K, N 91; transcript B, N 141.
1068 Transcript C, N 108.
1069 Transcript G, N 135.
1070 See, for example, discussion in Chapter 5 Section V.B.2.c.
1071 Transcript D, N 81; transcript K, N 91.
1072 Transcript K, N 91.

and financial resources are put into a project. As these investments are key for the developers, they believe that they should be legally recognized. According to this group, protection should start here, where all these investments are actually made.[1073] They argued for example that a program's prototype would represent the first moment in development at which a (partial) result could be shared with and exposed to third parties. Only a convincing concept with a significant sales potential is able to generate high economic value in the future.[1074] Consequently, investors and principals try to detect and select a potentially profitable project as early as possible, and participate in it before anybody else does.[1075] Elaborated concepts and prototypes were, consequently, seen as particularly interesting from an economic standpoint for potential business partners. But as soon as the program becomes noticeable for these, it also becomes accessible for competitors and, thus, is at risk of infringement. The interviewees felt that it would be reasonable and fair to set the starting point for legal protection at this stage, where the good reaches its highest economic potential and for the first time is also exposed to competitors. Another important point they stressed was that there are particular inventive developments that can be implemented in different ways.[1076] But it would not matter how the invention was realized because the specific idea behind the concept is what matters. If the software's main achievement lies in the vertical domain know-how that is visualized in a business procedure, the technical implementation is negligible.[1077] If only the final product was legally protected, the software in its earlier stages would be left unprotected, although society could make use of the computer program in its earlier testable versions, or the descriptions and technical documentation. Protecting software in its later stages would be too conservative. According to more than half of the interviewees, legal IP protection should start at conceptualization.

The rest of the interviewees argued for the final product to represent the starting point for legal protection. At this stage, the product becomes fully useable and fulfils its purpose.[1078] But the interviewees also felt that only when the software was finished could the full economic potential of a computer program be discovered. The product can be commercialized and put on the mar- 550

1073 See argument in transcript L, N 112.

1074 See transcript D, N 50 for examples.

1075 Transcript B, N 86; transcript D, N 34.

1076 Transcript A, N 160.

1077 See transcript L, N 112.

1078 Transcript I, N 159; transcript H, N 114 and 134; transcript F, N 125.

ket to generate revenue.[1079] At this point, the completed software product reaches its biggest economic value and can be exploited in the most damaging way. For certain types of creations, the key value lies in carefully thinking through the solving process in order to determine the optimal final shape of the good. For a sophisticated computer program that is able to analyse dynamic data, the concept's realization may be where the essential definitions are made and where the main contribution is provided, for example through efficiency. In this case, the good's final arrangement is therefore highly important for the final offer of the product.[1080] Thus there seems to be a group of software developments whose key contribution is found in a particularly creative or inventive technical realization. Where this is true, it is the technical implementation that increases the software's economic potential, makes it attractive for exploitation, and thus triggers the need for protection of the final product.

551 The interviewees agreed with the conclusion that it is not simple to determine a clear starting point that would do justice to all possible applications, but instead, a case-by-case evaluation is needed. This analysis would, according to the interviewees, have to take into consideration *when along the development axis the creative or inventive contribution actually happens*, be it a particular, elaborated conceptual thought or its technical implementation. However, no protection should be offered until the creative idea *has been sufficiently specified and fixed to overcome the critical step*.

552 Another problem not yet considered is that we often assume that the software development process works linearly although, in practice, more often companies work iteratively or with an incremental approach. Although linear and spiral development models show similar substages – ideation, conceptualization and realization – in non-linear approaches, the intervals between the different stages are shorter.[1081] The engineer returns to the concept phase several times, switching back and forth between conceptualization and realization. It is therefore particularly difficult to isolate a starting point in the iterative development approach. One interviewee suggested finding a separate approach to determine the starting point for legal protection for these development methods, for example working with *semi-manufactured or intermediate deliverables built in separate sprints*. Here, where each modular part of the program has

1079 See transcript I, N 159.

1080 Although only one interviewee discussed this chain of thought, I believe that it summarizes what many others tried to say with their examples (see transcript L, N 112).

1081 See for example discusson in transcript A, N 160 and 162, and transcript D, N 83.

gone through conceptualization and is minimally programmed, the section is preliminarily fixed.[1082] Before the end of the sprint, the individual section is still dynamic and adapted to newer findings. The basic idea of the iterative and incremental method approaches is precisely that the individual component, once completed, is no longer changed in its basic features. As many interviewees explained, after a section's sprint, the program module has become *viable* and able to stand alone. Only minor conceptual adaptations or technical subtleties are reserved so that the component at the end fits into the overall composition. At the same time, the intervals in iterative approaches are short enough so that third-party copying becomes highly unlikely; there is simply not enough time within a sprint to rebuild a single component. After a finished sprint, however, with the module's release, the individual component becomes exploitable by third parties and requires protection.

4. The End of the Term of Protection

The following section looks at which factors are important in deciding how long a computer program should be legally protected and the software companies were asked how long the term of protection should be for computer programs. 553

a) Points of Reference

Different terms of protection serve different functions of IP protection. In order to answer the question of how long computer programs should be protected, we first have to find factors that influence optimal duration of protection. I thus asked the interviewees what they considered as relevant in determining the term of protection. 554

In the interview series, three reference points were discussed: 555

To start with, a few of the interviewed parties believed that the maximum term of protection should be determined by the *will of the company or author* of the computer program. Commercializing and developing software is closely connected to the company, its brand and its customers.[1083] The product thus has a close relationship to the company that made it. On the other hand, every software developer takes high risks and makes big investments. According to 556

[1082] Transcript D, N 83.

[1083] Transcript L, N 108.

the interviewees, the companies that create the product, give it its shape and take the risk should also have the right to decide how long their product should be granted legal protection.[1084]

557 Some other interviewees suggested that the *utilization* of a computer program should function as a point of reference. They felt that a computer program should be protected as long as it was still purchased and used by customers.[1085] As soon as the number of users runs dry, the economic interest of competitors and, thus, the risk of infringement stops and the computer program no longer requires IP protection. This approach would be particularly useful for the evaluation of products with nostalgic values and/or evergreens. In this context, one of the interviewees referred to Microsoft and its Office package. These products have not lost their value although used for years in their original version.[1086] A similar reference was made to widely used Google products, such as their search engine, Youtube or the Android operating system. As long as these software products still work and the users keep on utilizing them, it was argued, the law should not hinder commercialization of a product that generates demand. The dynamic and continuous growth of computer programs based on old code should therefore influence the maximum term of protection.[1087]

558 The third and final suggestion, made by most interviewees, refers to the technical *usability* of computer programs, how long they can be applied technically. After some years, a technology becomes outdated and more difficult and less comfortable to apply. The developers as well as the users need to invest more and more effort in keeping the technology behind the product up and running. The software's value for the users decreases. At some point, the users might stop using the product entirely and replace it with a newer product.[1088] Competitors also become less interested in reproducing the product because they, too, would have to invest some effort to keep the outdated program interesting. When the product is no longer utilizable in an efficient way, the end of the product's life cycle is reached. According to most of the interviewees, the end of a product's usability should constitute the most important point of reference in determining the maximum term of protection.

1084 See argument in transcript C, N 114.

1085 See for example transcript I, N 110, transcript K, 89, transcript L, N 106, and transcript D, N 75.

1086 Transcript L, N 106.

1087 See transcript D, N 75.

1088 Transcript B, N 164, and transcript E, N 147.

The interviewees named three points of references to determine a computer program's maximum term of protection, including the ongoing utilization of the software by its users, the will of a program's author or owner, and the estimated term during which a software product stays technically useable. Most of them agreed that the last point, referred to here as the usability of the product, should be used to determine the maximum term of protection. 559

b) Adequate Protection Period

Having discovered which reference points can be used to determine the time length, the interviewees were asked what they believed was a reasonable and adequate maximum term of protection for computer programs. 560

Most interviewees believed that the computer program's usability should be considered in determining the maximum term of protection. The software market exhibits a particularly fast pace of product launching, but it is also quite short-lived. Life cycles of computer programs are said to be shorter than for other types of classical art works. The interviewees thus agreed that a modern IP framework should consider that the market periods of software are shorter than of other IP-protected goods. The current standard term of protection in copyright for computer programs of a minimum of 50 years after the last author's death was unanimously described as too long, as technology becomes outdated and loses its purpose long before the end of the term. Such a long term of protection would therefore only "foster the gains of possible inheritors" and facilitate "evergreening". However, today most software is developed and managed by legal entities, not individual natural persons. The heritability of copyright, including the long term of protection, thus loses its purpose.[1089] In addition, the interviewees believed that, following the original legitimation of IP rights in general, the term of protection should be short enough so that society is still able to make use of the product when it falls into the public domain. The artificial market advantage the right holders obtain should be just long enough so that the right holder can get established in the market: "Either the creator is able to regain his [or her] investments during the first couple of years, or he [or she] will never be able to do so".[1090] 561

The replies to the question of how long computer programs should be protected by law all lay somewhere between 10 and 20 years. Most were in favour of a time limit around 10 years as an adequate length of protection. They be- 562

1089 Transcript I, N 157; transcript F, 117.

1090 Transcript G, N 131.

lieved that after this time the program's usability suffered greatly and that, consequently, the enjoyment for the customers diminished. If the users' needs and the technical requirements in the field of application were particularly stable, the interviewees believed that a longer term of protection of up to 20 years could be justifiable. Stable user needs and technical requirements favoured longer usability. In the absence of these, the shorter term of protection would suffice to satisfy the developers' needs and practical desires.

563 The interviewees suggested a different approach to determine the term of protection for IP rights that included amplified exclusivity and particular market advantage, such as patents. The companies agreed that the main question in this problem should be how long a company should be allowed to "till the field" before competitors are allowed to enter the market.[1091] According to several interviewees, a couple of years is long enough to get established in the market and achieve a market advantage.[1092] If a product is not amortized after the first 10 years, "something must have gone wrong".[1093] The product would no longer be worthy of a legal market advantage. Upholding the long term of protection of patents no matter the merit would cause excessive negative effects on the market. Consequently, for more exclusive intellectual property rights, such as patents, a shorter term of protection between 3 and 10 years would be sufficient.

564 Some of the interviewees further highlighted that a legal protection system for software should take into account the fact that, due to modern development approaches, software is a continuously changing product.[1094] Successful products are rearranged and renovated again and again.[1095] One of the interviewees referred to Google Search as an example, which was established approximately 20 years ago. He said it was probable that parts of the original code lines were still in use in the current version.[1096] Most of the interviewed companies emphasized that the mere fact that some parts of source code are retained for more than 20 years would not automatically mean that it should be

1091 Transcript F, N 117.

1092 See transcript E, N 165 and 167.

1093 Transcript G, N 131.

1094 One interviewee described it by referring to a river: "Is it still the same water if I wait for fifteen minutes? With software, everything looks similar from the outside, but inside it keeps on changing. The question is, when does the time inside start to count again?" (transcript B, N 154).

1095 Transcript B, N 152.

1096 Transcript D, N 75.

granted prolonged protection. Still, the interviewed developers believed that the law should carefully consider today's technical possibilities and make a conscious decision whether or not this point should have an effect on the term of protection.

To conclude, the interviewed parties first discussed which reference points could be used to determine the protection period of computer programs. Most suggested that the expected period during which a computer program would still be technically applicable should be the main point of reference to determine the maximum term of protection. The interviewees further concluded that a maximum protection time of 10 to 20 years would be sufficient to provide the engineering companies with the protection they needed to earn back their investments. An even shorter time period of only 3 to 10 years was suggested for more exclusive IP rights on software, such as patents. A new time-limit framework would also have to consider that software development and its release has changed in the last two decades and that newer development models, such as the continuous delivery approach, should be reflected in the term of protection. 565

5. Setting the Limits: Admissible and Inadmissible Third-Party Interventions

Having determined what the protection scope of computer programs could look like, the next step is to establish where this scope is to be limited and to what extent third parties are allowed to use legally protected goods. This section does not discuss the effect of classic legal barriers but instead focuses on discovering some important limits to the scope of protection. It starts by discussing the borders between original and second-hand works, as well as the specific problems of source code translation. It then discusses how the so-called quality ladder affects software engineering, and where limits to the protective scope lie in the case of standards and enhancing inventions. Finally, the interviewees suggested how computer programs could be compared from a legal perspective. 566

a) Delimiting Second-Hand Works: Where Do they Offer a Contribution?

The interviewees explained that many parts in a software product would build on previous discoveries, either self-developed or from external sources. Similar to a quality ladder, existing parts are recycled to create something new and 567

better. Software engineers refer to it as "to fork".[1097] They confirmed that it was widely used in software development around the world and was also encouraged by the Open Source and Free Software movements.

568 With regard to the legal analysis of IP rights, several interviewees stated that they had great difficulties determining in their practical work when and to what extent they can use third-party programs as a spur, and when they are actually crossing the line to a third party's IP infringement. At the same time, they said it would be equally difficult for them, to conclude with certainty that the associated IP rights had been infringed for their creations. While they agreed that *apparent imitations* should be legally forbidden, the interviewees also highlighted how difficult it was to set the limits in blurry cases. *Similarities* between two products lie in the nature of things and are common: "Would you like it if a green coloured chair and a yellow coloured version of the same chair each were allowed to have an independent copyright?"[1098] Another interviewee emphasized: "A chair is a chair!"[1099] They said that the legal boundaries to determine when software "A" becomes too close to software "B" remain unclear.

569 Still, the interviewees were able to give some perspective. They had stated that a development should offer a creative contribution or added value to be granted legal protection,[1100] and repeated this for delimiting second-hand works. The person using the original work, that is, imitating the source code, has to provide a personal contribution to the outcome to be rewarded with their own legally protected intellectual property. For this purpose, the second-hand good needs to exceed a creative step with its additional contribution, otherwise, the secondary work would continue to legally depend on the original. The potential creative contribution of a second-hand good should therefore be considered when evaluating a derivative.

570 But, according to the interviewees, verifying whether a work represents a derivative from another does not end with comparing the two final products. Several of them emphasized that there might be some technical problems that could not be solved or implemented differently from the original. In this case a competitor cannot offer a new implementation but instead has to use similar

1097 See also definition of the term: Techtarget, "fork", available at <https://whatis.techtarget.com/definition/fork> (retrieved September 6, 2021).

1098 Transcript F, N 115.

1099 See transcript E, N 155.

1100 See above, N 454 ff. and N 462 ff. for more information.

concepts and solutions.[1101] Referring to the example above: After all, a chair is a chair!, the interviewees emphasized that the duplication of the one-and-only possible solution should not be qualified as a derivative work. They recommended that the law should take the developer's *creative leeway* into account and verify whether and to what extent two companies have utilized the same programming language, followed identical requirements, are dependent on third-party instructions, and to what degree the outcome of the two results is *exchangeable*. They believed that, if there was creative leeway, two entirely different outcomes would probably result because there would be different ways and more possibilities to realize the same idea and requirements, with different parameters or frameworks.[1102] However, the more detailed the requirements are, the narrower the creative leeway of the developer and the fewer possibilities he or she has to create something novel and original.

At the same time, the interviewees noted that, beyond the requirements, there would be some degree of *best practice* or *standards* within each area of software development. The developers have to follow these best practices, otherwise they risk the usability of their software decreasing. This problem can be illustrated by comparing car rental web pages. The online services usually provide similar methods to search for cars, to arrange the search results, etc. There are only a few additional functions which are not offered by competitors. The creative leeway of car rental web pages thus seems to be limited. The user is accustomed to a common look-and-feel and appearance. The interviewed parties believed that these best practices or standards should be taken into account when evaluating computer programs. They suggested that, instead, we should positively value distinctive characteristics, for example of a look-and-feel, that differ from the usual best practices. The expectations of originality or novelty, using the terms of copyright and patent law, therefore have to be adapted to the subject in order to avoid artificially narrowing the scope of application.[1103] Evaluating potential derivative works should thus include the possibility of predetermined developing factors and best practices. 571

To conclude, in evaluating second-hand works the creative and inventive contribution of a development should be considered. The examiner should observe the positive inputs of the secondary work and evaluate whether these are worthy of their own legal protection. In this evaluation, the examiner should also assess to what extent the developer had creative leeway in creat- 572

1101 See for example transcript C, N 44; transcript H, N 114 and 118.

1102 See transcript H, N 114; transcript K, N 31.

1103 See for example transcript F, N 115.

ing this second-hand work. They should also examine whether and to what extent a work is based on predetermined characteristics due to the use of a specific technology, and where the author of the derivative has overcome these specifications and basic requirements to offer a contribution of his or her own.

b) Translations of the Source Code in Particular

573 In most interviews, it was stressed repeatedly that the software system's environment was quite open to interventions from the outside.[1104] While many external interventions in software are already legally qualified as illicit editing of a work, it is highly debatable whether and to what extent translating the source code from one programming language into another represents a violation of authors' rights in the studied jurisdictions, and whether it can be factually detected. The interviewees expressed their concern about the current uncertain situation. They were strongly against lawyers differentiating between translations within linguistic languages and programming languages, as this distinction would represent discrimination of the digital literary expression.

574 They emphasized that the source code is dynamic and therefore highly adaptable.[1105] It can, if requested, be translated into any other programming language within the same group of languages. At the same time, technically developing the source code frequently represents the main creative contribution in a specific product. If the source code is copied and translated with a translator into another programming language, the program, although different in the literary expression, would still follow the same elaborated commands. The person copying the source code and entering it into a translator does not themself offer any personal contribution to the outcome, and thus does not pass the creative step for it to become a legally protected good of its own.[1106] The question to what extent translations of the source code are legally covered remains widely unsettled. For this reason, the software companies would want the limits of source code translations to be regulated explicitly, so that all legal uncertainties are eradicated.

1104 See above, N 401 ff.

1105 See previously N 401 ff.

1106 See transcript C, N 106

c) *Limiting Extensive Rights of Standard Essential Patents and Fostering Incremental Improvements*

One function of IP law is to foster innovation. At the same time, absolute and far-reaching exclusive rights are designed to exclude others from developing a similar invention or one building on it. For this reason, some people assert that intellectual property law tends to prevent innovation in technology rather than fostering it.[1107] In a field such as software, where a lot of development and novelty is still happening, exclusionary rights can have a considerable effect on the competitors as they can block others from researching and elaborating important basic developments. 575

According to the interviewed companies, it is important that law provides a specifically tailored and balanced solution for new technologies, such as software. It should provide an environment that is invention-friendly but not at the same time blocking others. Law tries to meet these needs and practical desires by allowing design around and, to some extent, decompilation in statutes. This has increased interoperability in software. However, many software companies believe that these measures do not go far enough. 576

aa) Standard Essential Developments

A lot of computer programs build on previous discoveries. For this reason, it was important for software engineers that standard essential patents and other protected knowledge of industrial standards were accessible and could be used through a facilitating mechanism. 577

One interviewee justified this need with an example from the mobile phone industry: "The first [mobile phones] had a poor screen, other ones a poor user interface, then the keyboard was bad. Each component was sheltered separately with patents, and nobody was allowed to make changes to them. And we, as the consumers, had to choose between all of those poor products."[1108] This example shows that many important developments are covered by absolute or exclusive rights. As described, they can have a blocking effect on 578

1107 In my research, a little more than half of the interviewees believed that patents were able to foster innovation in technology as this provides an incentive to invest in R&D. The others were more critical and emphasized that powerful patent holders prevent others from being innovative.

1108 Transcript A, N 142; see also the example described by the same company in transcript A, N 96.

people who have to develop software. Although the engineers can partially avoid implementing these technologies, the effort to do so comes with great expense and compromise. Other technologies cannot be circumvented because the end products have to comply with certain standards.

579 According to the interviewees, it is important that a reasonable scale is used in granting exclusive rights, especially for standard essential software components. One suggested solution for this problem was that right holders could be forced to grant access to standard essential developments. Some observed that this could involve a compelled grant to use against payment whenever a development was necessary to fulfil the usual work of an engineer. For this purpose, a specialized authority or other body would have to declare it essential to have a particular legally protected development as standard. The person entitled to the development would then have to partake in compelled licensing to the third party but would receive remuneration in return. In any case, better access to standard essential development should be enabled in order to maintain the innovation mechanism.

bb) Incremental Improvements

580 Another question is how law should deal with cases in which no standard essential technology is used, but, instead, a developer discovers an *incremental improvement* to an existing product or method. According to the interviewed parties, this incremental improvement could constitute a better, more efficient, more effective or more flexible solution to a known technical problem. They emphasized that, particularly in software engineering where a lot of inventions are still possible, the broad protection scope of patents would have a wide-ranging effect, making patenting as a whole subject to attack. Many improvements to pre-existing technical innovations, which could be of great use to society and the market, are blocked through the broad protection scope in patent law.[1109] It was questioned whether patents in these situations still serve their original purpose.

581 The interviewees argued that incremental improvements should clearly be legally protected, if the improvement, the positive gain for the existing innovation, would overcome an inventive step and would thus itself fulfil the legal protection requirements and obtain an absolute right. But if an improvement represented an incremental inventive gain for the previous state of the art, but

[1109] This issue was raised in several informal preliminary talks and discussions with software engineers and professors teaching in this field.

itself was not able to meet the legal requirements for its own absolute right because it was too dependent on the original innovation, they suggested it should be legally allowed, if the improvement was a distinctive one. If an absolute right impedes such improvements simply because the legal requirements are not met, the intellectual property right protecting the original development would be encroaching it. For this reason, the interviewees wanted a tailored solution to legalize incremental improvements in software engineering, for example with the help of compulsory licences or injunctions.

582 To conclude, the interviews showed that the engineers wish to gain admission to standard essential technologies, but the use of important basic developments is legally forbidden. In the disputed field of incremental improvements to previous innovations, in the absence of access the companies are denied (equal) opportunities. It was suggested that legal measures, similar to compulsory licences or injunctions, are implemented so that developers can make use of standard essential patents, and would be legally allowed to share their incremental findings with existing developments.

d) Second-Hand Works: How To Compare Computer Programs

583 In the legal foundation, above, it is described how law has found ways to cope with second-hand works and delimit them from originals.[1110] While case law has introduced comparison standards for most areas of creative works, the interviewees felt that legal practitioners and courts had shown great difficulty in applying the standard institutions to computer technology. Practising lawyers have called to my attention the problem that there are no guidelines on how to compare two software products, in order to evaluate a possible infringement. It is difficult for lay people to identify the similarities and differences in multilayered software products, and even more difficult to say when these similarities have become legally relevant. I therefore asked the interviewees to what extent and how they would differentiate between and compare similar software programs.

584 The interviewees acknowledged this problem. They agreed that, in the end, no tool or expert can guarantee discovering a copy or an inspiration theft. If somebody wants to hide that he or she imitated or copied, it remains difficult to prove.[1111] Although they believed that a technical expert would most probably be able to evaluate whether one work was (partially) developed on the basis

[1110] See above, N 365 f.

[1111] Transcript D, N 56.

of another,[1112] setting the limits between admissible inspiration and inadmissible imitation or copying would, in their opinion, be very difficult. No matter what good or service we are looking at, it serves a certain purpose and is used in a specific context. Instead of looking at the object of evaluation at a distance, the interviewees suggested looking at the different layers of software to tackle the problem: "When I look at the surface, both programs do the same. You may compare this to Microsoft Dynamics and SAP, two accounting software programs. At the end, you pay balances and invoices. So, from this perspective, you have identical situations. You may say that one software product completes its task in a particularly effective way, more steadily. I could focus on a level further down and note that the database design varies and that two different types of technologies are used (...)".[1113] Consequently, we can find different types of technologies, involved process designs and approaches to guide the user and compare different computer programs. By focusing on different sections or subdivisions of the observed object, it becomes easier to see the similarities and differences in a computer program.

585 The interviewees stated that we could focus on certain specialities and characteristics in order to compare two computer programs:

- customer processes: the way business processes are constructed and managed;[1114]
- particular features or the total extent of functions and features:[1115] one can observe how a computer program behaves and on which requirements it is based;[1116] the programming languages and technologies may vary but the key functions remain the same, if copied;[1117]
- the source code: similarly to literature, one can compare the written text and evaluate to what percentage two different source codes corre-

1112 Transcript D, N 54.
1113 Transcript L, N 96.
1114 Transcript L, N 98.
1115 Transcript E, N 157, transcript K, N 73, transcript B, N 110, and transcript C, N 88.
1116 Transcript K, N 73.
1117 See transcript B, N 109.

spond;[1118] specifics to compare include procedure identifiers and parameters,[1119] source code comments,[1120] the front-end, the architecture of the code,[1121] and the structure and building of the code;[1122]

- the user interface: visual perception is sufficient to determine strong similarity;[1123]
- the look-and-feel: how did the engineers implement the features and functions and how does the user experience a computer program;[1124]
- the database design;[1125]
- implemented technologies, such as computer language families and used algorithms;[1126] and
- provided documentation, notes and instructions.[1127]

All these characteristics and elements may be taken into account when comparing software products. For the source code alone, one can also use particular technical tools that complete the comparison, like the software used to compare computer programs with Open Source programs. But there are also tools that help to visualize the structure of the source code, which would also help to identify similarities between programs.[1128] 586

In this context, the interviewees emphasized that there were certain best practices or standard solutions in software development. For a lay person, these might be red-flagged as an undue similarity not offering any creative contribution. But these solutions would represent standards that were inevitable in practice, and thus frequently used and observable. To be able to technically compare two programs, it would be important to recognize and acknowledge the existence of these best practices. 587

To conclude, this section looked at which aspects an examiner could focus on to compare two software products. Whenever strong similarities in two pro- 588

1118 Transcript G, N 114, transcript I, N 123, transcript C, N 86, and transcript H, N 105.
1119 Transcript B, N 107.
1120 Transcript B, N 117.
1121 Transcript D, N 54.
1122 Transcript B, N 107.
1123 Transcript C, 84 and 86.
1124 Transcript K, N 73, transcript L, N 98, and transcript B, N 117.
1125 Transcript L, N 96.
1126 Transcript C, N 84, transcript L, N 96 and 100, and transcript I, N 114.
1127 Transcript B, N 117.
1128 See discussion in transcript D, N 56.

grams are observed, the examiner should take into account the existence of standards and best practices in developing a program. The examiner can thereby ensure that he or she does not red-flag predetermined characteristics in a product, and instead focuses on the relevant incidents.

VI. Law Enforcement: Infringements and Legal Disputes

589 Another function of intellectual property law is to regulate how unauthorized interventions in a right holder's scope of protection should be handled and the consequences for an infringer. Hereby, the legal entities get to know what rights and obligations they have. Having looked at how software is developed and commercialized, and which factors are important in building an IP law system that is consistent with software technology, we now explore how infringements affect software developing companies, and how and to what extent they can enforce their IP rights in case of an infringement.

A. Infringements

590 The software developers were first asked to what extent they had experienced an unlawful breach of their IP rights on computer programs.

591 Every party stated that they had observed some kind of incident concerning their IP rights. They described how it is difficult to evaluate whether such an incident can be considered an actual infringement of their rights or whether it was just a critical event.[1129] The scope and impact of the violations varied, but both contractual breaches and third-party infringements were described, although contractual infringements were said to happen more often.

592 According to the interviewees, in most cases clients had registered too many users to a licence-limited server, conducting so-called *overuse* of an established licence agreement.[1130] As the right holders of the software do not want to compromise the existing relationship with their customers, they usually just pointed out the excessive utilization to the infringing party, abstaining from further measures. The interviewees said that many of their customers would not notice if they infringed contractual terms with their actions. Customers generally did not do it on purpose, and thus were surprised to learn that they

1129 See for example the struggle of one interviewee in trying to explain what he considered as an infringement, in transcript H, N 142.

1130 This usually falls under contract law, see for example transcript I, N 6, transcript E, N 175, and transcript C, N 118.

had committed unlawful use of the software. If pointed out by the software licensor the customers would usually agree to pay the additional licence fees without any trouble. The right holders would try to react moderately and individually in these cases, as they would not want to damage the existing relationship with their customer. An informal note would usually suffice.[1131]

If an unrelated third-party conducts an infringement, stricter and more formal measures would be imposed.[1132] Because there is no existing relationship, and thus no trust bond, there is usually no reason to show restraint. In this situation, legal measures are brought much quicker. The differentiation between current clients and unknown third parties therefore determines a company's reaction to an infringement. 593

Infringements were consequently described as a relevant topic for right holders, into which they had to put quite a lot of care. It appeared that each had developed their own particular way of dealing with violations. Economic interests, for example behind a customer relationship, were shown to have a great effect on shaping their infringement policy. 594

B. Legal Disputes

The interviewees were then asked to describe their company's experience with legal disputes in the field of software. 595

They emphasized that legal certainty would be a very important factor for every researching, developing and commercializing company. Their property would have to be secured. A functioning system of law would be measured to some extent by whether it offers transparent, comprehensible and proportionate tools to enforce a right holder's rights. It would thus be important to provide a satisfactory litigation system for software problems. 596

Prior to preparing the interview guidelines, I had the impression that only very few software IP cases were brought to court in Switzerland. However, several interviewees described how they currently were or had been previously involved as a party in a legal dispute on one side or another. Very few companies stated that they had no experience at all with litigation.[1133] The companies mentioned that, in the past, they had been able to resolve most disputes directly with the opponent, and had not been forced to take legal measures. Al- 597

1131 Transcript L, N 114.

1132 Transcript C, N 118.

1133 See transcript H, N 144; transcript I, N 134.

though most of the litigious procedures had happened with third parties, most infringements were said to have occurred with an existing customer or licence relationship.

598 Being involved as a disputing party in litigation was described as tedious; it would hardly be a good experience because the litigants had already invested a lot of time and money to get to this step. In general, the companies said they preferred an out-of-court solution. The risks associated with a dispute could be tremendous.[1134] Litigation was described as a matter of *ultima ratio*. They explained that a direct talk usually had a positive effect. Only if an infringer acted with malicious intent or was unwilling to cooperate was going to court considered. Before a formal complaint is filed, all options, chances and risks are taken into account.[1135] Nobody wants to pick a quarrel if not absolutely necessary.

599 Several interviewees said that although they had been victims of an infringement, they had decided not to pursue it and institute legal proceedings. This was partly because the infringements were not severe.[1136] One stated that his company would accept infringements as long as they remained marginal. To some extent, they would consider them flattering.[1137] However, others explained that they considered legal software protection and the associated measures as a "toothless tiger",[1138] and they saw themselves in a rather defenceless position. Litigation in the field of computer programs was very complex and required a lot of resources.

600 Legal insecurities were said to deter small and middle-sized enterprises with less experience in legal confrontations from standing up for their interests and rights. They said that they had tried to learn rough guidelines, in particular for

1134 One of the interviewees stated that he would check the case ten times before even thinking of litigating (transcript B, N 185).

1135 For example, companies have taken into account the following circumstances: How complex is a case? Is the company well documented on the infringement and prepared for judicial measures and disputes? Do any ethical, moral or reputational matters interfere with litigating? What are potential retaliations? What are possible benefits and potential costs? What are the actual chances of winning and losing? How much time and other resources are required? Or is it necessary to take legal proceedings just to make a mark or achieve precedent? Do contracts with the opponent party contain jurisdiction agreements or arbitration clauses?

1136 See for example transcript K, N 96, and transcript L, N 122 and 126.

1137 Transcript K, N 93, and partially also transcript H, N 138.

1138 See particularly transcript B, N 170, and transcript L, N 140.

the U.S. Common Law jurisdiction, however it was difficult to comprehend the diverse and partially inconsistent case law.[1139] This lack of knowledge also meant that not enough preventive and defensive measures in case of potential disputes were taken. Some private institutions and associations offered help for companies with smaller budgets and start-ups, and intermediaries such as Apple and Google would jump in by assisting with legal contacts and dispute resolution mechanisms if the software was offered within their systems. The private parties, however, could not fulfil the needs of all smaller parties.[1140] A certain degree of independence from other market participants should be ensured. Affordable public services are therefore urgently needed.

601 The lack of resources and smaller budgets for litigation reduced the possibilities for the smaller and middle-sized enterprises to defend themselves properly, if sued. The costs of legally defending a rightful position were described as substantial. The issue of cost consequently could have a huge influence on defending rights. The subject of the high costs of legal disputes was addressed in almost every interview. The decisive factor would be who could hold out the longest. As software disputes are said to be particularly difficult and complex, the costs for experts, consultants and lawyers are tremendous.[1141] The problem of costly defence was said to be partially used in the market to drive out smaller businesses. If a small business becomes interesting enough and has reached a certain threshold to be noticed by larger competitors, but the small firm does not want to collaborate with the large company, litigation would be a means of getting rid of the small ones. As the small companies cannot afford the defence costs in the long run, the larger companies will drain the small business' financial resources.[1142] They will try to simply drive them out of business or acquire the patents at low cost in the event of the small business' bankruptcy. This lack of resources and preventive measures was considered to reduce the chances of a smaller company succeeding in a legal dispute, even if the smaller company was in the right. Enforcing their rights and defending themselves against patents would thus be a particular challenge for smaller companies. Even if successful, the effort needed to confront even a single in-

1139 See particularly transcript M, N 50.

1140 Transcript K, N 73.

1141 The interviewee in transcript G, N 167 ff. stated that they usually calculated EUR 1-2 mio. (estimated value) for procedures in Great Britain and the United States.

1142 See explanations in transcript E, N 121.

fringer was described as being too high compared with the possible final earnings.[1143] Litigation, even if on the passive side, would thus be an unattractive option.

602 The right holders consequently tried to avoid legal disputes as the effort and resources required were not worthwhile in relation to the potential profit. Litigation in the field of software engineering therefore is considered a measure of *ultimo ratio*, preventing many right holders from standing up for their rights.

VII. Summary

603 The interview series showed that several aspects have changed in software engineering over the last two decades. There were 34 key findings from my interview study, which I summarize here:

604 1. The relevance of software engineering has increased in the last few decades, as it has largely replaced mechanical governors and electronic circuits in technical processes. Software engineering has become more important for industrial manufacturing and in the provision of infrastructure and services. Nowadays, most companies rely on computer programs. As a process and a service, software engineering is thus valuable and important for many industries. [N 375-376]

605 2. Several phases can be observed in a software development process. During ideation, the inspiration, need or idea is first conceived. In conceptualization, the engineer concretizes the simple idea by creating first sketches and drafts. A more detailed concept may also consist of elaborate functions and features. Most of the resources are put into this phase. In the next step, realization, the individual functions, features and drafts are assembled. All the necessary programming is conducted. When the software product is finished, it is implemented in its software environment, and either installed on the customer's computer or provisioned on a server. Further maintenance or assistance may be necessary. [N 379]

606 3. The process of how software is developed has changed over the last decade. While in the early years software was developed linearly – from the idea, to the concept, to the fully coded software product – today most engineers work at least partly iteratively. This means that they circle regularly between the con-

[1143] See transcript K, N 93.

ceptualization and coding process, drafting and realizing module by module instead of coding everything at once. With the help of this approach, a prototype can be achieved much faster. The prototype, as a minimal viable software product, can then be presented to customers and potential investors. In practice, a mixture of both development approaches is usually followed. [N 380-381]

4. Another trend currently observed is for software to be continuously delivered to the customers. Instead of selling separate new major versions of a product every couple of years, the computer programs are managed on accessible servers, where the software can be constantly altered and newly released in shorter cycles. Entire new versions, add-ons and updates can be released on a regular basis, sometimes even monthly. This encourages the implementation of improvements and spreading them much faster. [N 382-383] 607

5. How software is developed will vary from company to company and usually depends on a cost–benefit analysis. While smaller companies tend to select a development approach based on the project they want to work on, larger companies usually work with standardized and predetermined project processes to achieve their goals. Companies also tend to collaborate more with external partners to purchase third-party know-how and expertise instead of working as closed shops. [N 378, 380, 384-385] 608

6. The development process – from the first idea to its final implementation – takes on average around three years to be completed. A prototype of iteratively developed software will take around two years to finish. The duration of a development process can vary considerably and depends on several factors, including the complexity of the technical problem and the required administrative measures. [N 387-392] 609

7. A programming language is mostly chosen for a specific project. The predominating factors in its selection are the pre-existing system environment, the skills of the responsible engineers, the support of the software community and ruling trends, and the personal preference of the engineer. [N 394-400] 610

8. Computer programs and their components are very adaptable to alterations and changes. Depending on the system surrounding the software, the available resources and the effort an engineer can spare, making modifications may either improve the system or bring it to its limits. A source code can easily be translated into another programming language of the same language group. [N 401-405] 611

612 9. More companies tend to collaborate and commercialize internationally today, making use of economies of scale and conquering new regional markets. Regulations and jurisdictions have a great effect on an enterprise's risk evaluation and may hinder it from expanding into specific areas. A uniform and consistent legal software protection framework would be considered helpful and desirable. [N 408-410]

613 10. Competition is experienced as an inspiring and empowering factor for market participants. Although many interviewed companies said that software patents in particular may have blocking effects on the software engineering market, none of them had ever been prevented by competitors from realizing and commercializing their software products. Alternative solutions for legal and financial obstacles could always be found. [N 411-417]

614 11. Licensing and providing programs as-a-service are the two most common price models currently used in software engineering. While licensing has been the predominant model for the last two decades, the newer online approach to offer software as-a-service is catching up rapidly and has even started to replace other price models. [N 418-420]

615 12. The influence of the Open Source and Free Software communities is noticeable in software engineering. Developing without these communities and the products they supply is hardly ever economical. The advantages of Open Source and Free Software include access to technical solutions, the regularly outstanding quality of the products for little or no money, the standardization they foster and the big community spirit that is shared among the supporters. However, a frequently mentioned risk of using Open Source and Free Software is the potential contaminating character of some of the software licences. It is a big challenge for every developing company to elaborate a clean and effective Open Source policy. [N 421-434]

616 13. The software life cycle – the time after a product has been implemented until it has to be replaced with a new software product – is around 10 years on average. It can vary between 5 and 20 years. Less successful computer programs may be taken off the market after only 2 years. The duration of a software life cycle depends on the particular circumstances. The longer a technology persists, the more expense and effort is needed to maintain and adjust the system to newer requirements. [N 435-440]

617 14. Knowledge in software engineering can be grouped into expertise, experience and vertical domain know-how. All three types of knowledge influence software engineering at various levels, including how the customer's needs are

addressed and difficulties in the technical implementation. The vertical domain know-how for the software market and involved processes is particularly valuable for software developers because it represents a key quality that is difficult to acquire through studying or work experience. Know-how exchange through collaboration, round tables and discussions is very important for all developing companies. On technical aspects, companies freely exchange experiences at conferences, on specialized platforms and within the Open Source and Free Software communities. For vertical domain know-how, however, companies instead rely on secrecy and contractually protected collaborations with third parties. [N 441-453]

15. Creativity is an important factor in software engineering. The term includes the way a problem is solved and a potential solution implemented. Being visible on both a business and a technical level, it affects every software development phase. Although the use of assisting services such as toolboxes and libraries is common nowadays, integrating these components into the remaining system environment remains a creative activity. [N 454-461] 618

16. The term innovation involves comprehensible added value for a third party and represents either novelty or progress from the initial position. Innovation in software engineering can be achieved in different ways. First, existing older inventions that were developed with mechanical governors or electronic circuits can be reintroduced digitally; second, entirely new inventions can be realized with the help of computer programs; and third, there is still large potential for inventions in the field of software engineering itself, including how software is created and maintained, as software engineering is still a rather young discipline. Inventions are imaginable for each phase in the development process, including new process management approaches and entirely new technical solutions. At the same time, most innovation occurs in the development phase of the project ideation, as it is the inspiration that contains most originative strength. [N 462-474; N 480-482] 619

17. In order to prevent trivial innovations from being legally protected, the expert evaluating the legal requirements should not be a person of ordinary skills, but instead show special skills in the particular science in question. If an expert with special skills in the particular area of software engineering (in which the invention is claimed) could not themself think of the inventor's way to solve the given problem within a reasonable time limit, an invention should be considered as non-obvious and therefore novel. [N 475-479] 620

18. All the interviewees agreed that some kind of legal protection for computer programs should be granted in the future. Without such legal protection, the 621

companies said they would continue producing software but would rely more on secrecy and would stop sharing their knowledge and innovations with the market and society. They would also make more use of technological security measures. [N 484-490]

622 19. Overall, there was high satisfaction with the current hybrid software protection system. The industry particularly acknowledged the possibility to copyright computer programs. Patenting is regarded as justified if the covered invention is truly new and original. Still, it was felt that some modifications and amendments to the current intellectual property protection system in the field of software would be desirable. [N 492-493]

623 20. Software developers believe that software has not yet been fully understood and accepted in intellectual property law. [N 497]

624 21. As the software industry shows a fast launching pace, the time factor is crucial in software engineering. The longer that administrative processes linger, the less a company can profit from early entry into a market. The companies hence would like shorter administrative processes for legal software protection. [N 498-499]

625 22. In the field of commissioned work, shared drafts and concepts were frequently copied without the principal having subjected him-/herself to any formal contractual obligation. There appears to be a lot of insecurity in the time leading up to a contract. Conceptual documents exchanged before the closing of an agreement should be better integrated into the legal protection system of copyright. [N 500-502]

626 23. Patenting trivial inventions has a damaging and preventive effect on the market. At the same time, qualified software inventions deserve to be legally protected, if they are truly new and original. Real software inventions go beyond a mere algorithm and include a technical effect or conjunction, a practical application area. For this reason, computers, too, fulfil the technicity requirement, as defined by the European Patent Convention for other types of inventions. For the interviewed software companies, the European Patent Convention's requirement that the program has to be implemented or connected to a hardware component is unnecessarily restricting. They trace back the current European patent practice for software to the legislator's lack of acceptance and trust in computer programs. [N 503-509]

627 24. It was the wish of most of the interviewed software companies that computer programs could be patented. However, the patent requirements should

be tested substantively in the application process in order to increase legal certainty of the right holders and competitors and to avoid patenting trivial developments. [N 504; N 510-511]

25. Often, only larger companies with more resources are able to supply the necessary means to protect and enforce their software with patents and copyright. The current diverse legislation represents a challenge for smaller companies and complicates rights clearing before a product is launched. Managing IP rights efficiently is difficult for smaller companies, which is why enforcing IP rights is of less importance for smaller businesses. At the same time, all software companies are afraid of patent trolls. The interviewed software companies said they would welcome preventive measures taken against potential excessive effects of software patents. [N 512-515] 628

26. A component is valuable to a software developing company if it represents a qualified creative or inventive contribution in the program it works in. Also the amount of knowledge and know-how, and the resources that were invested into its development affect the product's value. Typically, the source code, the algorithms, the combination of particular functions and features, the look-and-feel and the vertical domain know-how in a computer program represent the economically most valuable components in a computer program. The valuable parts of software, which exhibit the highest concentration of sensitive know-how, demand most of the resources for their development and also offer the greatest creative or inventive contribution should be granted legal protection. [N 526-544] 629

27. A development should be granted legal protection as soon as it has reached a minimum level of substance and maturity. For this purpose, the project idea should be specific and comprehensible enough so that a reader knows how to technically apply it. The moment when a program is fixed sufficiently to meet these requirements cannot be predetermined abstractly but has to be identified case-by-case. While developments that include a lot of know-how often manifest their particular contribution in conceptualization, developments that require more technical finesse in implementation are usually distinguished at a later stage of realization. In the case of the iterative development approach, a first prototype at the end of the first sprint may serve as the earliest point of legal protection. The concept needs to show a minimal degree of composition and an adequate force to be granted legal protection. [N 545-552] 630

28. The current protection period of 50 years after the last involved author has died was unanimously described as too long. Further, linking the term of protection to a programmer's death was considered as nonsensical. Software was 631

said to represent a merely economic good, which is why, instead, the protection period should be tied to the right holding company and, according to most software companies, the software's release date. Regarding a program's expected usability, a maximum protection time of 10 to 20 years after its release would suffice, according to the interviewed companies. An even shorter time period of only 3 to 10 years could be implemented for more exclusive and absolute IP rights, such as patents. New technical possibilities, including the possibility to continuously update and alter software on servers, makes it difficult to determine a term of protection. [N 553-565]

632 29. In practice it is often difficult to determine whether a third-party computer program has infringed one's rights. If a later development is able to exceed the creative step and offer an additional creative contribution, it should be rewarded with legal protection. In this evaluation, the creative leeway should also be taken into account, for example whether some predetermining technical or practical factors constrained the engineer's creativity. At the same time, if the later work does not provide this required contribution on its own, it should remain legally dependent on the earlier, original work. [N 567-572]

633 30. The source code can easily be translated into other programming languages within the same language group. Similar to translations in linguistic languages, translating a source code from one programming language into another should be explicitly legally forbidden, unless the output offers its own qualified contribution. Otherwise, the translation would simply rely on the same principles and creative implementation as outlined in the original. [N 573-574]

634 31. As software engineering is still a relatively new discipline, it is important for law to elaborate a tailored and balanced solution, which considers that, in this field, exclusive rights can have a particularly large blocking effect. For this purpose, standard essential technologies should be made available for everybody in exchange for fair remuneration. At the same time, incremental improvements to an existing product or method should be legally allowed if they offer a distinctive positive gain to the previous situation. [N 575-582]

635 32. For technical lay people it is difficult to compare software products, especially in terms of infringements. It was recommended that the technology, the involved process design and the way in which the user is guided should be compared to find similarities and differences. On the other hand, an examiner should take into account certain predetermining factors and best practices in the respective fields. [N 583-588]

33. All the interviewed companies had experienced some kind of infringement of their IP rights. Most infringements had occurred within existing contractual relationships, where clients had conducted overuse of the licence agreement. Infringements conducted by third parties were more seldom. These were usually countered with sharper measures than infringements of regular customers. [N 590-594] 636

34. Most of the interviewed software companies were or had been involved in a legal dispute over software IP. As legal disputes require a lot of resources, software companies try to build on alternative preventive measures in order to minimize legal disputes from the start. Consequently, legal disputes remain a matter of *ultima ratio*. Smaller businesses can hardly afford to be a party in legal disputes, and thus have difficulty in enforcing their IP rights and defending themselves in case of legal charges. [N 595-602] 637

Chapter 6: Discussion of Selected Problems

638 This chapter picks up what I consider as the most important findings within this thesis from the previous chapter, and connects them to the knowledge gained in the technical and legal foundation. It discusses to what degree the results correlate with the current legal understanding, and how we could use the findings to draft new approaches in law that better suit the current needs and practical wishes of the software development industry. Due to the limited frame of this thesis, the goal of this chapter is not to cover every possible angle or develop complete models. Instead, it aims to offer a legal classification of the findings by highlighting difficulties in the current legal system and pointing out areas of room for improvement.

I. Preface: Is there a Future for Legal Software Protection?

639 Software engineering as a scientific discipline has developed greatly during the last two decades. Through its achievements, it has obtained a significant meaning not only for engineering companies, but for all industrial sectors. One observation from the interview series is that most businesses today rely on computer technology in one form or another.[1144] This trend is still gaining weight as more industries replace mechanical governors and electronic circuits with digital information technologies to fulfil technical solutions in their inventions and works.[1145] Many mechanical processes that were carried out manually back in the 1980s today are partially or fully automated. Perelman described in 1985 that source code statements were initially written on paper, but then eventually typed into a computer later. This is unimaginable today.[1146] Nowadays, software has become the easiest and most effective way to develop, manage and work industrial processes. As a consequence, software today has a clearly established and unbreakable position in the manufacturing and service sectors.

1144 See N 375 f.

1145 See N 376; the particular economics of software engineering and the inexhaustable repeatability of a digital process are just some benefits that support the transformation of the industry; see N 189 ff. and further description FISHER, 16.

1146 PERELMAN, 922.

While the importance of computer programs has increased, their value is similarly growing. Boehm suggests that in IT hardware products it would generally be the integrated software that determines the value of the product and not vice versa.[1147] In the interviews, a significant and large hardware developer confirmed this statement.[1148] If this is true, it would mean a considerable change of paradigms in the industry, using analogue technology to make profits from digital ones, a shift that is worthy of recognition and should be reflected in our legal system. 640

The international community has tried to integrate such changes in practice by offering copyright for visual and literary expressions in computer programs, and, in some regions, full patent protection for inventions such as algorithms and digitally implemented procedures. The condemning verdict of the interview series makes it clear that engineering companies believe that the product "software" is still not fully captured or understood by society – and this was particularly true for the legislators and judicial powers that make the rules for software. The interviewed companies feel misunderstood and are frustrated.[1149] An assessment of the U.S. Congress's Office of Technology of 1990 shows a similar interpretation, stating that software protection, already back in 1990, was not able to plausibly integrate software into intellectual property law. According to the Congress's report, the difficulty of capturing computer technology in law is caused by three main factors:[1150] First, the *nature of software technology* is not really tangible for lay people. Second, there are major difficulties that emerge when *harmonizing social and economic differences* in regulation between the legal and technical communities. Third, it is hard for law-makers and for those applying it to comply with a *fast-moving and dynamic software market* which is characterized by the fast pace of product launches. These three arguments identified by the U.S. Congress are still applicable today. But there are further practical problems involved. According to the findings in the survey, software rarely has any lead time to second-comers in distribution, as functionally similar products can be offered rapidly once the original is on the market, even if the successors do not directly profit from an original source code. It suffices that a product and its technical solutions become examinable.[1151] Consequently, the *time factor* is of great importance for 641

1147 See Boehm (1981), 17 f.

1148 See transcript G, N 15.

1149 See N 497.

1150 See the official summary in U.S. Congress (1992), 3 f.; referring to U.S. Congress (1990).

1151 See particularly N 498; also described for example in Samuelson et al., 2364; Lemley (2002), 1890 f.

everything associated with software engineering, be it the time-to-market[1152] or the administrative formal procedures[1153] that are necessary to be granted a protective patent right. The law should reflect this circumstance and do it better justice. Otherwise it has little chance of keeping pace with new technological developments and loses its power to steer.[1154]

642 Another important issue is that the way software is developed and managed has changed in the last two decades. Today, a software product is usually *produced and distributed across national frontiers*. Various contract parties are involved on an international scale and a higher number of *inter-party collaborations* is noticeable.[1155] At the same time, the common software licence system has been partially replaced by *service models*.[1156] This allows companies to retain the valuable source code, and to provide the software on servers rather than sharing its hard copy with a customer. Companies also embed strong *technological security mechanisms* in their software that help to prevent abuse by third parties.[1157] These changes can be described as private security measures. While the use of such private preventive measures increases, in theory, legal mechanisms become partially – although not fully – dispensable.[1158] In this argument, the main function of intellectual property law is to assign rights rather than to enforce them in case of infringements. As the rate of services and server-distributed computer programs increases, users receive less access to the source code of software and, consequently, the number of infringements should decrease.

643 Like the chicken and the egg problem, the question remains: what came first? More technological measures that make a legal framework less important for preventing infringements, or a weak legal framework that presses private entities to work with other security measures than law to protect their software? According to observations made for this study, developers make use of private protection mechanisms to complement a legal system that does not fully pro-

1152 See N 198 f., N 498, N 641 and N 760.

1153 See N 498 f.; also discussed in U.S. Congress (1990), 9.

1154 See also discussion in Dreier (1999), 6 ff.

1155 See N 384 f. and N 408; see also discussion in Scotchmer (2006), 227 ff.

1156 See N 418 ff.

1157 See N 488 and N 674; there is a similar observation in: Lessig, 765, with further references; Thomann (1992), 5.

1158 Although copyright helps to enforce these technological security measures through sanctioning their circumvention of copyright; see Art. 11 WCT.

tect their assets or offers less effective measures to enforce their rights.[1159] Software developers have adapted to the current legal system and are, to a certain extent, capable of helping themselves. Still, the legal system does not seem perfectly balanced and efficient, if private entities have to carry the compensation for its malfunctioning on their own shoulders.

644 This study has shown where the potential problems in legal software protection lie. The question now is how to continue. First we have to recognize and acknowledge that all the interviewed software companies favoured the legal sheltering of computer programs. They appreciate the benefits it brings and made it clear that without legal protection, no socially acceptable solution could be achieved.[1160] On the contrary, without legal protection for software, they would in future develop their products without any public access, focus more on secrecy and embed even stronger technological security measures. Innovation would be fully privatized, not shared with society. Based on their arguments, we have to conclude that it has been right in the past and also would be reasonable in the future to protect computer programs under some type of intellectual property law.

645 At the same time, it became evident that the doctrinal differences in Europe and the United States cause a lot of confusion among software companies. The interviewees described how the legal uncertainties resulted in increased search and allocation costs, and discouraged software developers from taking the risk of expanding in the international market.[1161] Consequently, there is a call for international legal harmonization. Jurisdictional differences in handling software protection should be further reduced and possibly avoided altogether. Taking the lead from modern software law, it would thus be desirable to eradicate dogmatic differences in law and to find an international solution and application of the common institutions that reflects the strong international characteristics of software development and commercialization. The answer to the initial question is therefore: Yes, there is a future for legal software protection, and the path has to be to legally protect computer programs in a unified international way.

1159 See particularly transcript C, N 66, saying that the missing patent protection complicates the situation; see similarly transcript A, N 113 ff.; transcript F, 69 & 81; see also discussion in DREIER (1999), 6.

1160 See N 484 ff. and N 492 f.

1161 See particularly N 408 f. and 507 ff.

646 But what should this legal framework of software protection look like? In the past, legislation has often reacted to a problem by applying the established structures to the new development. In protecting software under the existing IP law framework, the structures of two traditional models were simply expanded with very few customizations. Scholars have devalued this handling as 'mal-suiting', 'a half-baked cake' or just "wrong".[1162] The need for a *sui generis* framework for IP protection of computer programs has been expressed from different corners. The main point of criticism is that knowledge-based industries, such as the software market, show different practicalities and therefore also different needs and practical requests from artistic ones, and that these are only loosely met by the modular and generic legal IP frameworks. A tailor-made framework for software IP rights would have different possibilities to integrate the current standards in software development, instead of following a 'one-size-fits-all' principle to define the scope, the term of protection and the remedy model a legal system provides.[1163] A tailored framework for software protection would enable finding perfectly adapted and more competitive solutions for a dynamic industry with its own particular needs and wishes. While I personally agree with this opinion, we should not forget that IP law is highly legally harmonized on an international level with an established framework of contractually agreed standards and treaties that also apply to computer programs.[1164] This makes a paradigm change politically difficult to implement. Further, special treatments in law should always be well founded and clearly in-

1162 See HARISON, 33, 75 and 113; ARMSTRONG, 196 f.; DAVID, 50 and 52; SAMUELSON ET AL., 2348 ff.; SAMUELSON (2017a), 1218 f.; REICHMAN, 2481; FISHER, 14; SCOTCHMER (2006), 83 f.; LUTZ, 653; HILTY/GEIGER (2011), 187; HILTY/GEIGER (2015), 616 f. describe the assigning of computer programs to copyright as arbitrary; the difficulties are further recognized in U.S. CONGRESS (1990), 1; SCHUHMACHER, 200 ff.

1163 BURKERT/HETTICH/THOUVENIN, 49 f. and 57 f.; SAMUELSON ET AL., 2365; LEMLEY/BURK, 96 ff; ALLISON/LEMLEY/SCHWARTZ, 1074f.; HILTY/GEIGER (2011), 188 ff.; BELL/PARCHOMOVSKY, in particular 232; HARISON, 193; REICHMAN, 2481 ff.; SCOTCHMER (2006), 117 f.; ZIRN, 206 ff; Samuelson et al. went so far as to say that the status quo has become so comfortable that nobody want to change it. But this would risk the stability of the legal system (SAMUELSON ET AL., 2365 f.).

1164 For more information, see 'International Context' in Chapter 4 Section III.; see also BOECKER, 33; referring to: the decision of OLG Frankfurt of July 21, 1983, 6 U 16/83 – *Donkey Kong Junior I*, published in GRUR, 1983, 757 ff.; decision of the OLG Koblenz of August 13, 1981, 6 U 294/80 – *Nutzungsrecht des Arbeitnehmers an Computerprogrammes des Arbeitgebers*, BB 1983, 992 ff.; decision of the German Federal Labour Court of September 13, 1983, 3 AZR 371/83 – *Statikprogramme*, published in GRUR, 1984, 429 ff.; decision of the OLG Frankfurt of August 4, 1983, 6 U 19/83 – *Donkey Kong Junior II*, GRUR 1984, 509 ff.

duced or necessary. Creating a *sui generis* framework for software protection is only one viable path. The other possibility would be to continue working with the IP system that we have today, and to make some minor adjustments and amendments to it in the governing regulation. This approach would recognize the globally established legal structures that we know and have, while better meeting the needs and desires of the software industry. Further, although the basis for IP law is provided and sustained in the international treaties, smaller adaptations to the current system could partly be implemented at a national level by national legislators, allowing some flexibility and individual locational advantages of proactive legal systems.[1165]

At the beginning of this study, I assumed that the benefits of a *sui generis* approach would outweigh the second possibility of making only smaller adjustments because a system change would be reasonable and more efficient. However, the interview series has clearly revealed that applying the present IP law system to computer programs would be generally acceptable for software developing companies and that the current approach is able to meet their most important basic needs.[1166] Although I understand the opinion of Samuelson et al. and others that a *sui generis* framework might be the best way to meet all the potential needs and wishes of the software developers and managers,[1167] the findings of the interviews do not sustain the conclusion that a tailored solution has to be packed into a legal framework of its own. A complete system change to a *sui generis* model is, at least within the scope of the present limited research, not necessary, as the current hybrid structures of intellectual property law suffice to serve as the basic structures for software protection. 647

Still, the software companies are calling for some modifications and amendments. These are necessary to meet some of the needs and desires as well as to reflect the practicalities that predominate in the software market. As the interview series showed, the software engineering reality and its interpretation in law are not compatible. The WIPO recognized this problem back in 1978, suggesting a set of model provisions on computer protection.[1168] However, most of the suggestions never found their way into the international IP treaties. So the need for change persists. Adaptations in the current legal framework could either target current smaller problems specifically for computer programs, or use the insights obtained from this study for a greater sys- 648

1165 Boecker, 36; Fisher, 14.

1166 See N 492 f.

1167 Samuelson et al., 2365 f.

1168 WIPO Model Provisions on the Protection of Computer Software.

tematic change to address creative works and inventions on the whole. The German professor Rossnagel in 1992 offered the term 'jurisprudential technology assessment', believing that law has to exercise its duty and take the chance to actively shape attractive socially, legally and environmentally compatible technical solutions, and not just react to problems by fire-fighting.[1169] Burkert, Hettich & Thouvenin further stated that the constant changes in the manifestations of information technology require ever new discussions on technology-related law.[1170] In my opinion, if law wants to be proactive, it has to provide a clear direction and not only acknowledge but embrace the needs and practices as well as the practical wishes of the software industry. This can in part be achieved through a contemporary reinterpretation of the available IP law. A more progressive way to modify software protection would be to install further steering mechanisms that demonstrate what actions the legislator considers desirable, and which activities are frowned upon. This is particularly relevant where we want to close certain loopholes and in cases where we want to set limits on the exclusive rights of the right holders, especially regarding IP law trolling.

649 The following discussion gives further clarification on what modifications in the current IP law system, constituted particularly of copyright and patent law for computer programs, could look like in detail; how continuing with the current system could suffice; to what extent we should improve it; and where explicit new solutions are required.

II. Copyright and Patent Protection: Hybrid Model or Copyright only?

650 Although the interviewed software companies on the whole confirmed preferring the current IP model for software protection over a *sui generis* approach, it represents an entirely different question whether we should continue with the present hybrid protection model – a composition of copyright, patent law, design law and mechanisms from legal institutions outside the range of IP – or switch to a one-approach model. Again, the subsequent discussion focuses on the two main IP rights in software protection: copyright and patent law.

651 The findings of the interview study showed that software companies particularly value copyright as a legal mechanism to protect their goods. It gives them

1169 Rossnagel, 36 ff.

1170 Burkert/Hettich/Thouvenin, 49.

the basis they require to assign and trade their products and services. They particularly appreciate the copyright's copy guard and its reference to the author.[1171] Applying copyright to computer programs therefore remains undisputed. The institute of copyright should hence remain the primary mechanism for protecting computer programs.

While copyrighting is frequently and well appreciated among software developing and commercializing companies, patenting finds itself in a more difficult position. The protection of software under patent law is probably one of the most disputed legal issues of our time. While in the U.S., the legislators suggest tightening the eligibility of software patents because of their potential for abuse, in European patent law merely digital programs without a physical embodiment are, at least according to the wording of the law, largely excluded from patent protection.[1172] These differences in the regulatory premises are reflected in the number of registered patents; the U.S. share of computer patents in the international patent class GO6F considerably exceeds the European ones.[1173] At the same time, the number of registered European inventions in this patent class suggests that, despite the exclusion in the European Patent Convention, some patents for computer programs are granted in practice, although not in the same quantity as in the United States. Different European patent offices and high courts have accepted computer-implemented developments as patentable, if they fulfil specific additional requirements. The patent authorities and courts hence use their room for discretion to protect certain software inventions. The exclusion of computer patents in the European Patent Convention is thus partially circumvented. But if even the different European Patent Offices apply the patent requirements in a manner that contradicts the apparently clear wording of the European Patent Convention, why has the law not been adapted? While every political attempt in legislation to change the system is suffocated, the patent offices and courts instead create and use loopholes or 'work arounds'. 652

The patentability of software has been and will remain a matter of great dispute. The main reproach in the literature and politics against software patents is that major commercial developers of software have been using their patent portfolio too extensively and that patents are in general considered "bad" or counter-productive for free competition. Both are claimed to be particularly 653

1171 See N 522.

1172 See Art. 52 para. 2 lit. c EPC.

1173 See Perspective STI Working Paper, 32.

dangerous for the field of software engineering.[1174] Similar statements were also made in the interview series.[1175] A large number of the interviewed software developers claimed to have been hindered in using particular supplies under exclusionary rights, in one way or another.[1176] However, all the interviewees gave the same fundamental statement that they had never been actually prevented from developing their software products in practice. If a particular component they wanted to embed was unavailable (or blocked), it could be replaced with a third-party offer or a created 'work around' without any greater effort. The market was therefore able to play freely, although some impeding effects were experienced. The negative effects of patents in the software industry were consequently seen to be limited and manageable. On the other hand, it was emphasized that in the Age of the Internet it would be difficult to survive and provide ideas on the market without obtaining exclusionary rights, patents in particular.[1177] Although patents do not guarantee full protection of the developed products and services, they shelter the broader scope of the software's implementation, offering the required lead time as well as a defence mechanism for the right holders.[1178]

654 Also in the interview series, patenting software was strongly debated. The two sides in the revision processes in Europe and the United States were also represented among the interviewed companies. However, to my surprise, the companies indicated that *patenting in the field of software should be eligible, if the claimed invention was truly inventive*.[1179] It seemed that patenting software was not a question of whether it should be allowed, but rather about the tolerated scope and assigned rights. It was frequently mentioned that granting trivial patents would endanger free competition and hinder the development and distribution of genuinely inventive creations, because the right holders got the chance to block third parties with a monopoly-like right they did not

1174 See discussion in U.S. Congress (1992), 23 and 135 f.; WIPO Standing Committee on the Law of Patents, 4 ff.; Mathematical Programming Society; Zirn, 149 f.; Melullis, 346 f.; Schwarz/Kruspig, 37 ff.; Walker, 54 f. See also a comprehensive study in Thouvenin (2005), 478 ff.

1175 See N 503 ff. and N 512 ff.

1176 For example because the supplier did not want to share his or her invention or because it was offered at too high a price.

1177 See N 411 f.; see also Merges, 579.

1178 See N 411 f.; see also discussion in Straub (2002), N 12; Scotchmer (2006), 85; Calame (2006), 656 ff.; Calame (2006), 404 f.; Melullis, 346 f.

1179 See N 509 f.

deserve.[1180] To this extent, the claims of relevant authors in the ICT law field would therefore be approved.[1181] At the same time, good software solutions prevail quickly, and as everybody wants to build his or her development on the basis of the newest technical solutions, a patent blocking them may also hold back other creative inventions that might follow. The next, more efficient, solution to a problem could consequently be obstructed. It could be said that this difficulty applies for every type of invention, not just software inventions. However, as computer technology and programs are increasingly replacing regular steering systems as well as analogue electronic engineering, this issue may indeed find new dimensions in the field of software engineering.[1182] I consequently agree with the conclusion that particularly in the field of software inventions a rather strict scope of protection would be favourable. Although the full extent to which patents affect the software industry today remains unclear, based on the findings of the interview series we can, however, confirm that excluding computer programs statutorily from the patent scope does not represent an appropriate solution to the problem for the software companies.

In international patent law, we can see a political patchwork of individual rules. 655
Already the regulatory bases contradict each other: The TRIPS Agreement states in Art. 27 para. 1 that member states have to grant patent protection for every type of technical invention, if an invention fulfils the classic patent criteria (novelty, inventive action and industrial applicability). The way computer-implemented innovations are invented (not developed!) today does in essence does not differ significantly from the invention process of the "seed drill".[1183] Excluding software from patent protection due to the type of technology it applies is therefore discriminatory and unreasonable. Following the clear wording of the TRIPS Agreement, software components should likewise be eligible for patent protection if they exhibit an inventive step. As Jaenich highlights, it is questionable whether Art. 52 para. 2 lit. c EPC conforms at all with the member states' obligations under the TRIPS Agreement.[1184] The European legislator has recognized the positive contribution of software inventions as well as the statutory difficulties associated with them, and has tried to achieve a regulatory change. However, all movements to resolve this issue have

1180 See N 475 ff., N 508 and N 512 ff.

1181 See: Cohen/Lemley, 39 and 50 ff.; Hilty (2018), 1186 f.; Arrow, 226 f.

1182 For example, for lawyers of the digital generation, Zirn holds that the classification of computer and software technology as non-technical is incomprehensible (Zirn, 174 f.).

1183 Merges, 586; see also discussion in Walker, 37 ff.

1184 Jaenich, 488.

failed.[1185] As a result, we have an unpredictable and insecure situation for software developers, companies commercializing computer programs, and also the authorities that have to evaluate the patentability of computer programs.

656 Overlooking the fact, that computer programs are generally excluded from the patent scope in Europe, the European practice focuses mainly on the technicity criterion to evaluate software-related inventions. Software is regarded as patentable if a further *technical effect* can be observed.[1186] The software companies stated that this practice is confusing because almost every software product goes beyond a simple mathematical algorithm and can therefore result in a technical effect or conjunction.[1187] The crucial question is whether the incorporated teaching is also novel and non-obvious. Arguing technicity for patenting computer programs thus represents circular reasoning.[1188] As identified by the U.S. Congress, one of the main difficulties with software is that people show great difficulties in fully capturing it.[1189] This makes it all the more difficult to apply time-honoured and established institutes such as copyright and patent law to it. Merges in his work described the same problem for the former U.S. patent system, back when it was very restrictive about patenting non-mechanical inventions: "For Thomas Jefferson (…), if you put technology in a bag and shook it, it would make some noise."[1190] This illustration perfectly explains what problem we have with software today; we cannot observe the (digital) movements, nor touch and physically experience its final product. We cannot put 'a piece of software' in a bag, and if we shook it, it would never make a noise. Asking for this would mean we would be asking for an additional criterion, e.g. a sound. The same is true for the current practice in Europe. According to the software companies, asking for an additional technical effect in computer programs suggests that the people evaluating software for patent offices and courts do not truly understand software as a technology, or focus

1185 The Commission proposal COM(2002) 92 was declined by the European Parliament on July 6, 2005, with 648 against 14 votes (see Out-Law.com, with further comment).

1186 See above N 280 ff., N 289 ff. in particular.

1187 Transcript M, N 39; above N 654 ff.

1188 Companies in this study that were familiar with patent applications in Europe argued that with the current legal interpretation, patenting software would not be a question of inventive creations but rather of the better argument for a technical effect (transcript F, N 90 f.); see also discussion in: Melullis, 341 f. and 350; Calame (2006), 661 ff.; Zirn, 174 f.; Schwarz/Kruspig, 34 f.; Jaenich, 488 f.; Walker, 57 ff.

1189 U.S. Congress (1992), 4 f., partially referring to U.S. Congress (1990).

1190 Merges, 585.

on the wrong factors:[1191] They seem to wait for the 'clashing', and thus overlook the technical concept or conjunction the computer-implemented invention has to offer. Similarly, the UK Court of Appeal established in its 2008 ruling that the general technicity criterion is fulfilled if an invention tries to solve a technical problem with natural forces.[1192] Whether this happens with a computer program or any other type of engineering should not matter.[1193] It is absurd for the European patent authorities to search for further technicity beyond the 'normal interactions'[1194] between a computer program and a computer. This was said to be frustrating for the companies concerned and caused emotional reactions that were also visible in the interview series. When law is not comprehensible, it becomes unforeseeable and starts to lose acceptance. This is problematic for the software industry.[1195] Bill Gates has summarized the software companies' perspective by saying that the European patent system is a rag rug that is very risky for innovators and software developers.[1196] The risk he refers to, from the perspective of an internationally operating group, may partly be due to the fact that the subject matter, and in particular the technicity criterion, is assessed differently in patent law throughout the world, with a particularly strict system in Europe. This was outlined by a comparative study of the European, U.S. and Japanese patent authorities.[1197] The global association BSA, the software alliance, advocates accounting for new innovations, emphasizing that governments should provide modernized laws that protect innovation regardless of the format or means of delivery.[1198] Tying the patentability of software to a technical effect therefore seems to be an unsuitable way of defining the subject matter of patents for software. In line with the findings of the interview study, the computer patent exclusion in Art. 52 para. 2 lit. c EPC should be struck out of the law.

1191 See N 508.

1192 *Symbian Ltd. v. Comptroller General of Patents*, decision of the EWCA of October 8, 2008 (EWCA 1066).

1193 Supported by: Staempfli Commentary to the Swiss PatG/EPC, Art. 52 N 55; Reichman, 2480 f.; Samuelson et al., 2331 f.; Schoelch, 271; Harison, 167 and 176 f.

1194 Decision of the EPO Board of Appeal of July 1, 1998 (EPO T 1173/97) – *Computer program Product/IBM*; decision of the EPO Board of Appeal of February 4, 1999 (EPO T 0935/97) – *Computer program Product/IBM*.

1195 See for example Thouvenin (2007), 664; Merges; Calame (2006), 450 f.; Wickihalder, 585 f.

1196 Comment by Bill Gates during a Microsoft Innovation Day in Europe, in: Boecker, 35.

1197 Patent Offices Comparative Study (1997), no 3.2. and 4.

1198 BSA-The Software Alliance, Global Software Survey, 15.

657 As previously outlined, the problem of extensive patenting and the dangers connected with it, including the blocking of evolving innovations, can to a certain degree be resolved by determining a reliable strict scope for software in patent law,[1199] starting with the definition of the subject matter. But how should the subject matter of patent law be circumscribed so that software can be integrated accordingly? Following the approach advocated in this thesis we do not need any additional or new definitions specifically for software patents, but instead can rely on the classic patent criteria. According to the classic definition, a technical invention involves a task and a solution or teaching in the form of a repeatable success or a rule that applies and manages the natural forces.[1200] Of course the invention has to involve a problem and either offer a solution or a teaching for it. The software method or product has to offer 'something in addition', just as it would not suffice to simply present a mechanical governor that did not complete a technical task or offer a technical teaching. Without the task and its solution, neither a software good nor any mechanical one would meet the technicity requirement of using natural forces to solve a problem and could thus not fall under the patent's subject matter. Hence, asking for a technical effect is either redundant, or, again, the policy makers implemented an additional criterion in the patent evaluation that is not contained in the international statutes. The problem of software patenting can be partly resolved by consistently applying the existing patenting criteria and examining the content individually and substantively, independent of the question of whether the object under review is a computer-implemented invention or not. Patenting of software should thus be enabled.

658 Based on these findings, continuing with a hybrid model of copyright and patent law therefore seems reasonable and, as such, appropriate to reflect the software companies' legal needs.

III. The Protection Scope in Copyright and Patent Law

659 If we continued with a hybrid model made of copyright and patent law, the question remains how we could harmonize the two institutions to better integrate computer programs as well as the processes involved to create and commercialize it; how could we define the scope so that the boundaries between the two become clearer? In the following section, the potential subject matter

1199 See above N 654.

1200 See above N 273 ff.

and scope of patent law and copyright are further evaluated and circumscribed, in order to then discuss which parts of a software product may in practice be subsumed under which institute.

A. Circumscribing the Potential Subject Matter

Patent law and copyright aim to protect inventive and creative contributions. Based on this understanding, the following section focuses on what should be sheltered under software copyright and patent law, and why such protection is necessary to protect inventive and creative works. 660

1. Discovering Creativitiy and Inventiveness in Software Engineering

a) Creativity in Software Engineering

Only creative works are protected under copyright law. The degree of creativity a work contains further determines its originality under the law. Creativity is thus of great importance both for a work's existence and for its differentiation from other works. According to the findings, creativity determines, in software engineering, how a problem is solved and how a solution is implemented. It involves artistic questions, but also smart solutions for business or technical problems.[1201] As Rossnagel explains for technology in general, but is also true for software, it is a product that is created and designed by humans.[1202] Software development consequently is closely connected to the practical circumstances under which it is formed and to the people that work on it. Creativity therefore is the result of human play.[1203] There is room or leeway for creativity where the scope for productive decision-making is out of the ordinary and the personality of the author can have a stylistic effect.[1204] The smaller the set of available solutions within a certain technical environment or ecosystem, the more restricted the discretion of the software developer.[1205] Creativity has to be searched for in every work individually. Analysis of the individual case hence remains necessary. There are, however certain indicators that help in determining whether something is creative or not from 661

1201 See N 454 ff. and N 458 ff.; see also Cross, 88 ff.

1202 Rossnagel, 63.

1203 Rossnagel, 68 f.

1204 See N 454 ff. and N 458 ff.

1205 See also discussion above, N 396.

the perspective of software engineering. As Agrawal et al. suggest, factors such as the degree of novelty, performance or attractiveness of a particular artifact as seen by the target (value), surprising or uncommon elements and combinations, future influence or relevance, coherence, semantic and logical appropriateness of a particular artifact (correctness), and comprehensibility may improve the scope or may entail creativity.[1206]

662 Creativity is visible in different ways, and can be offered in various types of contributions within a software product. As explained, it starts with finding a creative solution to a problem and continues with how an engineer plans to realize this idea, as well as how he or she wants to implement this solution technically. To decide on how an idea should be presented involves many different decisions, based on careful consideration and planning. It is important to recognize that there are most often several ways to implement and express an idea; there is rarely only one possible solution. The developer is the one who makes the important decisions. These are often influenced by personal taste. A part of the developer's character is reflected in his or her work. Similarly to how the brushwork and style of painters varies, the preferences and the way of designing and realizing a project in software development does too. Skills are not decisive in themselves, but experience and know-how influence the development style followed.[1207]

663 At the same time, creativity is reflected in different stages of software development and affected by several aspects along the way. As represented in the present thesis, there are three main phases in software development: ideation, conceptualization and realization.[1208] Creativity influences each development phase in a different way. It starts during planning, where the careful decisions determine how a product will be structured and organized, and how it can be presented neatly. Next, a suitable programming language has to be selected that is able to fulfil the technical requirements and gives the software engineer room to realize a project according to his or her plan and style. Choosing a programming language is predetermined by particular factors. It has to be ap-

1206 AGRAWAL ET AL., 2 ff.

1207 See N 441 ff.; see also AGRAWAL ET AL., 2.

1208 See N 158 ff. and N 378 ff.

plicable in the pre-existing technical environment[1209] and the engineer has to possess the skills to work with this language. In order to realize the product within a reasonable schedule, it is also wise to use a programming language that is well supported in the community so that the engineer can make use of existing elements. But, most of all, it is a matter of personal taste and preferences.[1210] How the instructions look is also a creative process. After all, it is the human's creative mind that writes the linguistic commands. These commands consist of parameters that are personally defined by the engineer and literary structures that are individual for each developer. The same is true for the visual elements in a software product.

Software solutions and systems have grown in complexity. According to Schmidt, more and more engineers have to elaborate their own particular solutions, as existing language mask functions are not evolving fast enough to meet all "next-generation" problems.[1211] Also if an engineer works with existing solutions, these masks have to be manually adapted to the individual development environment. This involves elaborating it with additional effort and time engagement. Where this adaptation process goes beyond technical functionalities, and there is room for decision-making on how to implement it, space for creativity is provided.[1212] Also the output of engineering-assisting tools may 664

[1209] See N 396; this commonly involves identifying the essential conditions under which the program will operate and making a clear decision on which tasks the software has to perform (Samuelson et al., 2328; Agrawal et al., 2). The technical environment in the form of technical and organizational components therefore is taken into account before a new project is started (Koreimann, 10 ff., 13 in particular; Wittmer, 106 f.). If an existing computer program is further expanded, refined or advanced, the engineer has to consider that the network environment has already been determined and consequently follows particular characteristics and uses existing conjunctions. If the whole previously existing software is built on a particular family of programming languages and all the parameters have already been defined it is pointless to embed a component that follows entirely different rules. Otherwise, the links to all the other elements and components won't work properly (see also further information in Koreimann, 10 ff., 12 in particular; Wittmer, 106 f.).

[1210] See N 394 ff. and N 399 in particular.

[1211] Schmidt, 26.

[1212] See N 454 ff.; see also: Koreimann, 231 ff. with regard to structured design and scope for decision-making; Rauber (1988), 23 f.; Contu, 55 f., 67 and 82.

leave room for creative work if the developer can make certain noticeable decisions in the creation of the tool, which then shape how the tool produces the outcome.[1213]

b) *Inventiveness in Software Engineering*

665 Only (novel) inventions are patentable. Still, the term invention constitutes a legal concept open to interpretation. But what does the *term invention* entail? According to the software companies, if from the perspective of an initial prior position, the presented process or good represents progress or incremental novelty, an invention is present. As patent law gives exclusive rights to the right holders, the enhancement should also bring added value for a third party such as society or the market, in order to 'earn' exclusive protection.[1214]

666 For the purposes of this thesis, the invention term should further be placed in the context of software. In 1899, the U.S. Post Office Commissioner, Charles Duell stated self-assuredly that "everything that can be invented has been invented".[1215] Although this statement turned out to be wrong in hindsight, it has been adopted for software time and again, claiming that everything that could be solved with software in an inventive way has been done and that further inventions in software engineering are impossible. It seems as if this debate is too focused on the term software and too little on the concept of innovation. Based on the findings of the interview study, inventiveness is visible differently in software development:

667 First, software can be used to develop completely *new inventions*, as in other technological fields.[1216]

668 Second, older inventions that were developed with mechanical governors or electronic circuits *can be reintroduced digitally*.[1217] This field of utilization will presumably particularly concern application and process claims. But transferring an analogue or mechanical innovation into a digital sphere – so-called digitalization or automation – won't suffice on its own to reach the level of an invention, if no inventive theory is introduced in addition to its transferral. The

---

1213 See above N 360; see also STRAUB (2011), N 67; Cherpillod emphasizes that working with existing parts may also lead to a creation (Staempfli Commentary to the Swiss CopA (Cherpillod), Art. 2 N 9); NEFF/ARN, 158 f.

1214 See N 463 ff.; see also discussion in: BELL/PARCHOMOVSKY, 234 and 277.

1215 Quote from MORGAN/LANFORD.

1216 See N 471.

1217 See N 471.

digitalization may, however, show inventive characteristics, for example by making a process more efficient, increasing its performance through reducing the number of inquiries or secure data transmission, or by making it in some other way more attractive. The solved problem further needs to be technical.[1218] These types of creations may be referred to as software-related inventions.

Third, as software development is still a rather *new scientific technical field*, the way in which software is developed and data is processed opens another potential field of application,[1219] expected to be particularly important for the field of process patents.[1220] 669

As Schwarz & Kruspig explain, computer science is a basic science. In contrast to other fields of invention, inventions can be implemented in both software and hardware. It offers applications in everyday items as well as utilizations in specialized areas, mostly in its computer-implemented form, in combination with other disciplines such as biology, physics and chemistry.[1221] Focusing on the concept of innovation, software engineering has opened a completely new field of science and is still significantly developing. Software consequently entails immense inventive potential. As software offers completely new processes and models, the treasure is limitless. Further, inventiveness can also be used during different phases of software development. Although usually the inventive peak occurs when the initial impulse for a new idea is given, working on the original idea during later stages helps to concretize it and put it into a particular expression.[1222] 670

2. Protected Interests

a) Know-How and Resources

Knowing what should be understood by the terms creativity or creative contribution, and inventiveness and inventive contribution, and to what degree 671

1218 This interpretation corresponds with the practice of the European patent office. See for example: decision of the EPO Board of Appeal of September 29, 2006 (EPO T 959/03) – *International Translatinos/Ed Pool*; see similar thoughts in BERESFORD, N 2.29 f.

1219 See N 472.

1220 DIJKSTRA, 209; MERGES, 586; FLOYD, 456.

1221 SCHWARZ/KRUSPIG, 34 f.; see also ALLISON/LEMLEY/SCHWARTZ, 1084 f.

1222 See N 480 ff.

both characteristics are found in software development, we need to figure out what would justify results that exhibit creativity and inventiveness being legally protected

672 Intellectual property laws were built to protect investment and provide an adequate incentive for creators to develop creative and inventive contributions, and share them with society.[1223] Pursuant to the findings of this study, creative and inventive processes in *software engineering require a lot of human, time and financial resources*. There are, on the one hand, the financial investments a company puts into the development of computer programs, and the time, labour and skill efforts spent on the other.[1224] The developers require technical expertise, experience and domain know-how to find plausible solutions to various problems and to develop a product the user wants to acquire.[1225] According to the software developers, domain know-how is the most valuable asset they possess.[1226] It incorporates the capabilities and particularities as to how a technical problem is solved with engineering, how to address the customer's needs and to recognize difficulties in implementing the solution.[1227] As the particularity of a development is a company's contribution, it is very important to keep and defend its know-how.

673 The difficulty with software lies in the fact that this *know-how and the invested resources are then easily accessible* from the outside once the software is published. The aim of the source code is to present the software in a literary form that is understandable and editable for developers. It is, by design, easy to manage and maintain as it represents the software's most accessible level. For the developers to alter it, it has to contain all the commands and instructions that control the behaviour of the software. It integrates all the data to conduct specific processes or offer particular functionalities, but also to visually illustrate the software in the form of the user interface. If not actively disguised, all the solutions, rules and approaches are also accessible from the outside for third parties. In this context, it should be remembered that developing software is not straightforward; the developing, testing, failing and succeeding comes from many hours of trial-and-error to find the optimal solution to a specific problem and implement it into a creation.[1228] The domain know-how

1223 See N 234 ff. and N 519 ff.

1224 See N 486 f., N 520 and N 535 ff.

1225 See N 441 ff.

1226 See N 443.

1227 See N 442.

1228 See N 142, N 423, N 444, N 529 and N 541; see the same conclusion in WILTGEN/GOEL, 1.

and resources that were invested to find this optimal, creative solution are then reflected in the source code and its visual implementation in the user interface. Suddenly the end product of time-consuming and cost-intensive development phases becomes accessible and traceable through observation of behaviour and reading the source code. This opens the risk of infringement, and publishing the software with a readable 'open' source code also makes it easier for third parties to imitate the product within a shorter time and at a much lower cost without direct copying, with the final problem-solving approach visible.[1229] As Harison et al. note, even if the imitation is not exactly the same, the functionality and features may be reproduced without extensive effort.[1230]

If the know-how of a software product is copied or imitated, the investments made and the future of the company are at risk. The software companies therefore try to *secure their know-how* as well as possible. If they have to publish their achievements or share them, they only do so on a contractual basis and with selected partners.[1231] As the technical means to copy and paste software have become better, software has become even more vulnerable. The developers have recognized this difficulty and try to hide and additionally secure their know-how with technological measures. These *measures are used complementary to whatever is provided legally in order to protect their products*. However, real security can rarely be achieved. Rueesch believes that the software industry has not yet been able to implement a lasting technological security barrier.[1232] Regarding the fact that all the interviewed software developers' IP rights had been infringed,[1233] it is legitimate to say that Rueesch's observation is true.[1234] Software companies thus have good reason to request that the sensitive and disclosed know-how worthy of protection, and also the time, money and human labour resources invested into developing the software, should not be exposed without adequate protection in IP law. 674

1229 See N 142, N 423, N 444, N 529 and N 541.

1230 HARISON, 174; SCHWABACH, xv; SAMUELSON ET AL., 2337 ff., 2366, 2409 and 2418; LUECK, 35; KOEHLER, 32 f.; MELULLIS, 343; THOMANN (1992), 4; SCHWARZ/KRUSPIG, 34 f.; LEMLEY/BURK, 91 and 92; LEMLEY (2002), 1892.

1231 See N 448 ff.

1232 See for example RUEESCH, 23 ff.

1233 See N 590 ff.

1234 See also N 584.

b) Know-How and Resource Protection in Copyright and Patent Law

675 The question is whether we can protect these interests appropriately with copyright and patent law.

676 Based on the two aspects we want to shelter – know-how and resources – we should continue with a two-fold protection model. The *hybrid model* of copyright and patent law protection makes practical sense as, due to their different objectives, the two protection institutions largely complement each other in the area of software: Patent law is conceptualized particularly for the protection of specialized domain know-how that is reflected in the inventive technical ideas and technical implementation. It protects the technical teaching or solution that is disclosed with an invention's registration, including the applicant's applied know-how. Copyright on the other hand protects the resources and work expenditure that was made and can be easily imitated, now that it is clear what a successful and working version of the component could look like. While patenting protects the software's disclosed know-how in the form of an applicable functionality, copyright aims for the manifestation of creative and labor-intensive work in an expression.

677 The two models together as a hybrid are able to cover the complexity of software, as a work that exhibits a lot of know-how on the surface and requires a lot of financial, time and human resources, that can be easily imitated at a lower cost and within a shorter time than when it was first developed. The combination of the two would also represent an adequate incentive to take the business risk and share their discoveries. The subject software including its component products can be well protected with the hybrid of copyright and patent law.

B. The Protection Requirements in Copyright and Patent Law

678 In order to be protected under copyright or patent law, we first need a subject that falls under the eligible subject matter and, second, the subject has to fulfil certain protection requirements. The protection requirements of patent law and copyright have been fixed through international codification in various multinational treaties, and are therefore in principle non-dynamic, they cannot be changed without a greater international renovation effort. In patent law an invention has to be novel, non-obvious and industrially applicable to fall

under the patent scope.[1235] On the other side, copyright law requires that the creative work represents an intellectual creation that shows originality or individuality.[1236] These protection requirements circumscribe the scope of protection in copyright and patent law irrespective of the object of protection. In practice, these fixed criteria have increasingly raised the question of the extent to which they can be adequately applied to "software" as a potential object of protection.[1237]

679 The software companies in this study said that they understood the outlined copyright requirements and would generally be satisfied with their practical interpretation.[1238] Similarly, the patent requirements were said to be consistent. Although they are not easy to interpret for lay people, the requirements would make reasonable connections and ask for the right prerequisites.[1239] *There would hence be no need to change the copyright and patenting requirements as such.* Nevertheless, it was shown in the interview study that the practical interpretation of the patent requirements could be adapted in order to meet the current challenges of the software industry better.

680 First, it was found that the software companies had a stricter interpretation of the potential *expert term* relevant for the non-obviousness criterion in patent law. The expert figure has been hypothetically based on an 'average' skilled person with the capability to think logically.[1240] Intuition or ingenuity is not expected of the expert, nor does the expert need to know and review all the prior art in the field.[1241] In the interview study, it was suggested that the expert should not only be a person of 'ordinary skills', but should also show special skills in the particular science in question.[1242] As Thomann explains, "the assessment of a creation presupposes mastery of the appropriate lan-

1235 See N 310 ff.

1236 See N 359 f. and 361 ff.

1237 See discussions in: Boecker, 35; Ohly, 809; Schulze 997 ff.; Harison, 113 f. and 193; economic analysis in Merges, 603 ff., see particularly 606; difficulties also addressed in U.S. Congress (1990), 5; difficulties also addressed in U.S. Congress (1992), 3; Samuelson et al., 2347 ff.; Bell/Parchomovsky, 232 f. in particular; Hilty/Geiger (2015), 616 f.; Straub (2001b), particularly 2; Lemley (2017), 909 and 943 ff.

1238 See N 493.

1239 See N 493; see similar discussion in Lemley/Burk, 95 f., concluding that the current patent criteria did suffice but would require active interpretation and reinterpretation.

1240 See N 316 ff.

1241 BGE 123 III 485, c. 2b; BGer of May 18, 2005, 4C.52/2005, published in sic!, 2005, 825; see also Staempfli Commentary to the Swiss PatG/EPC, Art. 1 N 80.

1242 See N 476 ff.; also discussed in Merges, 598.

guage (source code, characters, notation)".[1243] Without this specific prior knowledge, an expert's evaluation is neither reliable nor assessable in terms of the available and used discretion. The expert should thus be a person that offers in-depth expertise of the particular field and also has practical experience with the standards and prerequesites of the industry. By raising the skills expected from an expert, the quality of the examination automatically increases. Only if the examination is conducted professionally is the examiner able to distinguish truly inventive from trivial patents. The quality of the issued patents consequently improves.

681 In addition, the software companies suggested for the *non-obviousness test* that the expert should also closely review other approaches in the technical field in question, apart from the solution offered. According to the software companies, only technical solutions to problems that set themselves apart from what the specialized expert themself would know and do in practice should be eligible for patent protection.[1244] For this purpose, extended involvement and engagement of the expert with the technical problem is required. *An invention should only be non-obvious if an expert with special skills in the particular art of software engineering when given the same task could not him- or herself think of the way in which the inventor solved the problem within a reasonable time limit.*[1245] Similar to the European *problem-and-solution* test and the U.S. *functional approach*, the expert would begin their review by establishing the objective technical problem they want to solve. But instead of simply establishing the closest previous art for the problem, the expert should dive into the technical field and try to evaluate the problem for themself. The starting point would be the inventor's patent application which contains a description of the problem he or she wants to solve and the revelation of his/her approach. The expert should just look at the problem the patent application wants to solve and try to establish what potential solutions he or she would have thought of within a reasonable time limit, if he/she *had* to solve the problem. The expert would not actually have to solve the problem; it would suffice if they examined the problem and sketched out some potential procedures. The expert would thereby automatically establish and evaluate which standards, prerequisites and known approaches in the prior art were relevant for the potential solutions they were thinking about. In a final step, the expert would look at the solution suggested by the applicant to compare their own

1243 THOMANN (1998), 9 f., discussing evaluating an author's original contribution in copyright.

1244 See N 463 ff.

1245 See N 475 ff.

solution ideas with that of the applicant. The expert would then assess whether he or she could have thought of the suggested solution to the problem within a reasonable time limit, and to what degree the applicant's solution was determined by external factors or already included in prior art. The expert should thus not only look at the final solution and evaluate whether the solution was obvious from the perspective of prior art, but also apply his or her own expertise. Only solutions that were distinctively different from the expert's suggestions to solve the problem within a reasonable time limit would be declared novel and non-obvious. Thus, the expectations of the inventive step would be increased, taking technical feasibility and practical problems into account.

682 The software companies also outlined that it was currently difficult to find prior art that could thwart trivial patents.[1246] A comparative study of the European, U.S. and Japanese patent authorities confirmed this concern.[1247] The suggested different interpretation of the non-obviousness term would enable the determination of approaches that may not necessarily be exhibited in prior art, but were themselves evident or a logical consequence of prior developments. The problem with finding such prior art would thus be reduced and the quality of the inventions granted protection would be increased. One criticism of this proposed approach might be that it would provide too much discretion for examiners, which entails certain risks. However, as in legal court practice, the existence of discretion in an assessment does not itself represent a problem. The state-recognized and authorized expert examiners are in a better position to recognize and reject a trivial creation, due to their high credibility. Only decisions that were obviously incompatible with the discretionary scope would be questioned, and would thus also be legally contestable. It therefore represents a manageable risk. Secondly, one could argue that with this approach there would be an increased risk of hindsight bias because the expert might be tempted to say that he or she would have found the same idea, looking at the solution proposed in the application. But because the expert is instructed to first think through the process, find a solution of their own and only then look at the suggested solution, hindsight bias could be prevented. On the other hand, the risk of hindsight bias and subjectivity always applies if another person examines a third party's invention, but the risk may be lower if a higher-

1246 See N 476; see same conclusion in LEMLEY ET AL., 168.

1247 Patent Offices Comparative Study (1997), no. 3.1., particularly regarding computer-implemented inventions; Patent Offices Comparative Study (2000), 28, particularly regarding business methods; see also discussion in HORNS, 13; SCOTCHMER (2006), 74 f.; HILTY/GEIGER (2011), 183 f.; ZIRN, 185 ff.

skilled expert evaluates the inventive character instead of an ordinary skilled one. This is because a trained person with special skills and experience in a subject would potentially know better what they were looking for. In order to limit the possibility of hindsight bias, I recommend that the expert term be altered as suggested. Putting both propositions together, the full explanation for the non-obvious test might therefore be: *An invention is non-obvious, and therefore novel, if an expert with special skills in the particular art of software engineering (in which the invention is claimed) could not themself think of the way to solve the given problem, as the inventor has, within a reasonable time limit.* If necessary, the process could be a double-track examination carried out by two experts. One expert could examine the application according to the current classic problem-and-solution or functional approach and evaluate the patent application while a second expert would try to solve the same problem on their own within an agreed time limit. This second evaluation could then serve as the basis to establish prior art. This suggested change to the procedure could easily be integrated into the current law without requiring any revisionary act.

683 Finally, the software companies suggested that an innovation not only needs to involve an applicable teaching but this teaching should also be useful to the market or society.[1248] *The software companies thus wanted added value to be included in the invention.* The question is how or where this usefulness criterion could be implemented in the current patent scope. One possibility would be to create an additional separate criterion. However, this approach would be laborious and require special support from the international community. A better way would be to embed it either as a tested aspect under the subject matter or in one of the existing patent requirements. The required invention could, for example, be integrated into the evaluation of the *subject matter*. A teaching would consequently only be considered as a teaching if it offered an inventive and useful theory. The teaching term would thus be interpreted differently. Only teachings that showed a potential benefit would be eligible under the subject matter. But the usefulness aspect could also be examined under the *applicability criterion*, specifying that a teaching was only applicable if it could be applied in a way that generated more value for society or the market, and not only the patentee alone. Both approaches could be implemented under the current patent regime without formal changes. I would personally favour its implementation under the subject matter of an inventive teaching, as this should be the main reference point and if the added value was not visible in

[1248] See N 465 ff.

the teaching the requirements would not have to be evaluated at all. This approach would also indicate the importance of the minimum quality a patentable object should exhibit, and would therefore be a suitable measure to tighten the patent scope.

All the other criteria could be interpreted as before. Each of the three suggested changes to the current interpretation would help to narrow the patent scope. This would increase the quality of the patent evaluation's outcome and enable trivial patents to be recognized more easily. These measures would also help to strengthen the credibility of software patents overall. But these measures can only have an effect if the inventions are *fully tested* before they are granted. The interviewed companies believed that an inventor or right holder should only be rewarded with a patent if he or she could substantiate the quality of their invention. For this reason, the substance of the subject matter and patent requirements would have to be fully assessed. In order to ensure that only truly inventive creations were protected, the software companies said they would even be willing to accept a longer application process.[1249] Today, how extensively the substance of a patent application is examined varies considerably between the different jurisdictions. In Switzerland, only the subject matter is evaluated, not the patent requirements. In the European Patent Office and in the United States Patent and Trademark Office, a full substantive examination is made.[1250] It would be desirable to standardize the evaluation procedures of the different patent offices in order to increase the legal certainty of both the right holders and the third parties. 684

C. Potential Subjects of Legal Software Protection

Part of the reason for this study was to figure out what was most valuable in software development and commercialization from the perspective of the software companies – their 'crown jewels' – and to what extent these assets were covered by IP protection. The following section discusses how different elements can already be protected under copyright and patent law and where additional protection is needed. Although this thesis wants to offer guidelines for future assessments, it is important not to generalize about which objects are covered under copyright and which under patent law. I therefore follow Wittmer's advice that every component of the software product should be 685

1249 See N 511.

1250 See N 333.

analysed in its individual combination and entity to evaluate whether it is worth sheltering and to what extent it falls under legal protection.[1251] The following evaluation thus functions as an illustrative example for the argument.

686 In general, each inventive component in a computer program can be granted patent protection and every creative expression within a component can be protected with copyright, if it fulfils the corresponding protection requirements of copyright and patent law. The decisive factor for its eligibility for legal protection, therefore, is not what kind of component it is but rather what it has to offer, what inventive or creative contribution it entails. For this reason, the potential objects should be tested individually. Usually, it is only some elements or components of a complete software product that are worthy of protection. The fact that some elements in a software product do not offer an inventive or creative step, and are thus not protectable under copyright or patent law, does not prevent the software from being protected under some type of intellectual property right. However, in this case, only the element that is able to fulfil the protection requirements of the respective legal frame is legally protected. On the other hand, if a component or a part of it offers less originality, copyright may still shelter it from being copied one-to-one, as the component in its particular expression, that is, the combination of letters or signs, may be statistically unique.[1252]

687 It should also be taken into account that it is not the type of software that is important, but the individual product. In principle, macros, program libraries, database management systems and computer games are also eligible for software protection.

688 According to the software developers and companies commercializing computer programs, the software components that contain the most sensitive know-how and require the most resources to develop are the source code, the algorithm, the look-and-feel, the visual user interface and the functions and features a product offers.[1253] Together with the vertical domain know-how, which may be applied in any work, these elements also represent the most

1251 Wittmer, 136.

1252 The theory of statistical uniqueness (Theorie der statistischen Einmaligkeit) examines whether a work is individual enough so that it seems highly unlikely that the same or substantially the same work would be created by a third party for the same work. Only if both questions are negative, is the work considered original (Kummer, 30 ff., 44 ff., 47 ff., 63 ff., 80, see particularly 67; BGE 134 III 166, c. 2.5; BGE 130 III 168, c. 4.1-4.4.; see also brief summary in Straub (2011), N 73 ff., with further references). See in detail N 363.

1253 See N 526 ff.

valuable parts in a developed good.[1254] The following analysis therefore starts with the testing of these named components for their copyright and patent law eligibility. The current copyright law also protects the whole software product against one-to-one copying, which is why this matter is also discussed. Finally, the copyrightability of the development documentation is a topic that is currently highly disputed among scholars, and this will be addressed in a final section.

1. Whole Software Product

According to past interpretation, the entire software product distributed on a medium or made available for download is protected under copyright law.[1255] The software companies described how software piracy, such as hacker attacks and unauthorized reproduction of sold products, represents an ongoing threat to the industry, considering the existing technical possibilities and the insufficient measures to enforce IP rights in some countries of the world today.[1256] 689

The final product is the combination of all achievements. It is the product that is offered to the client and that is purchased for its potential use. It is also what the user in the end wants to apply. Single components may have their worth, but only if everything developed is combined, can it reach its full potential. The software product therefore represents the culmination of everything created and invested in the software development. If the product can be used without paying for it, it loses its full value. At the same time, the person who has a copy of the software can analyse its details, and thus access the know-how and creative solutions applied to build it, without participating in the redemption for the invested resources. As the compilation of components and mini-programs itself cannot be qualified as an inventive technical achievement but rather represents a creative decision, and because the interests at stake are mainly the invested resources and creative skills, it would appear reasonable to continue protecting the product under copyright. The whole product may also be protected against having undue advantage taken of somebody else's achievement under unfair competition law. 690

1254 See N 527 f.

1255 See N 356 bullet point 1.

1256 See for example N 537.

2. Code

691 The *source code* represents one of the two forms of a literary instruction that, once translated by a compiler, tells the machine what it has to do. As it is designed in a human-readable form, it works like a text with digits and letters so that engineers who are experienced in working with and 'reading' programming languages can understand it. The source code follows the logic that the engineer made it follow. Apart from quite functional aspects, it therefore also contains creative inputs of the engineer and a rationale based on his or her personal know-how and experience. In its final shape, the source code, as Perelman says, represents the engineer's "unique expression of a computer solution to a particular problem".[1257] Its construction usually requires time resources as an effective source code line has to be written and rewritten several times until the desired command is formulated.[1258] This is true for the whole source code, as well as for smaller sections of just a few combinations or a very distinctive code line within the source code. Due to its high creative quality and the amount of resources that its development requires, the source code is worthy of legal protection if it represents an original contribution. It should be noted that, as the code has to be preserved in a readable form, all its achievements, including the know-how used to build it, are 'served on a plate'.[1259] This was also recognized by the international community, which decided to protect the literary form of computer programs, and especially the source code, with copyright.[1260] As the source code does not itself constitute an invention, but rather represents one particular creative form of expressing it, it should fall under subject matter of copyright.[1261]

692 Not just the instruction itself but also the way the source code is organized, its so-called *architecture*, deserves our attention. The distinctive parameters and the compilation of instructions are structured into sequences, which, as a creative contribution, often earn copyright protection. All these parts within the source code represent creative merit that is dependent on the expertise and style of an engineer, and thus it goes beyond a mere technical formulating of

1257 Perelman, 923; see also Ernst, 209;

1258 See N 142, N 423, N 444, N 529 and N 541 and N 673.

1259 See particularly N 529.

1260 See N 259 for the TRIPS Agreement, N 262 for the WCT, see N 356 bullet point 2 for current interpretation.

1261 See also Harison, 175 f.; Melullis, 342; Samuelson (2012), 159.

instructions. The code is the result of how these instructions were put together within the predefined frame. A particularly distinctive organization and sequence of the source code can therefore also be copyrightable.[1262]

The *machine code* contains the program's instruction in a form that the computer understands and is able to process. It is therefore the reflection of the source code in another literary form. Although it cannot be directly read or analysed by humans, it can be decompiled in order to obtain a program's functionality,[1263] and thus a relevant part of the creative solution to a problem.[1264] The decompiled version of the machine code represents an object that can be easily harvested. In order to offer a technically appropriate framework of protection, it would be reasonable to integrate this with the know-how and creative solutions, placing the machine code also under copyright protection. This interpretation corresponds with that of the international community, which also shelters the machine code as a literary expression of a copyrighted computer program.[1265] 693

In all three forms of literary expression, the question remains to what extent the copyright protection applies. Again, in a case-by-case analysis, we have to determine which components or overall compositions of the source or object code exhibit originality. In practice, courts and private entities frequently make use of quantitative analyses to verify copyright infringements whenever they are suspected. However, as Straub explains, these analyses do not permit direct statements on originality and can therefore only make a statement as to whether, in the case of a high degree of accordance, there is a higher probability of infringement. At the same time, a high degree of accordance may also indicate that tools and components provided in the software community had been integrated into the software.[1266] This does not mean that no creative leeway was used to embed these elements. A quantitative analysis may therefore only work if it is combined with a subsequent assessment of a work's originality on a case-by-case basis. 694

1262 See N 356 bullet point 3 for current interpretation.

1263 See N 137, N 141 ff. and N 356; see the same reasoning in Commentary to the German UrhG (Loewenheim/Spindler), § 69a N 5.

1264 See N 142, N 423, N 444, N 529 and N 541 and N 673; for legal perception, see also Straub (2003); Staffelbach; Samuelson (2012), 159.

1265 See N 259 for the TRIPS Agreement, N 262 for the WCT, see N 356 bullet point 2 for current interpretation.

1266 See for whole abstract: Straub (2011), N 86 f.

695 Finally, the source code and the machine code, but not the architecture of a source code, may also be protected against having undue advantage taken of somebody else's achievement under unfair competition law.

3. Algorithm

696 The algorithm tells the computer in a functional rule how to behave and what to do. As the software's behaviour determines the service it offers, the algorithm that defines this behaviour represents a very valuable part of a software product.[1267] In order to obtain an algorithm that displays a general rule, a developer has to apply his or her know-how, expertise, constructivist thinking and verbal capacities.[1268] As the algorithm itself represents a technical idea, and is not understood as an expression, it is commonly excluded from copyright protection and instead put under patent protection, with some restrictions.[1269] The patentability of algorithms, however, comes with a conflict of objectives. On the one hand, the algorithm contains most of the know-how in a software product and should, thus, generally fall under the set of legally protected components. On the other hand, as Boecker explains, the behaviour of a computer program often reflects a general idea, whose use by the public should not be prevented unnecessarily by patents.[1270] Protecting basic algorithms endangers the freedom of ideas and entails negative effects, such as blocking other developers from integrating them and from entering the market.[1271] It is therefore important to distinguish when speaking of algorithms,[1272] and to find a way to filter inventive algorithms from basic ones, and exclude the latter type from patenting. For this purpose, I have compiled a simple testing procedure to demarcate basic algorithms from more refined ones on the

1267 See N 145, N 527 and N 539; see also the statements in: Harison, 183; Samuelson et al., 2318; Boecker, 130, concluding from Haberstumpf (1983), 222.

1268 See N 145, N 473, N 527 and N 539; Hommel et al., 37.

1269 See N 356 bullet point 5.

1270 Boecker, 140 f.

1271 See discussion in N 411 ff., N 503 ff. and N 653 f.

1272 As Straub emphasizes, the term algorithm as such is not in itself suitable for assessing the patentability of computer-related inventions. A distinction must be made between abstract mathematical principles that need to be legally accessible and concrete technical solutions (Straub [2011], N 445).

basis of the patent requirements and the results obtained from the interview series.[1273] The procedure complies with the criteria as stipulated in the ruling of *Aerotel Ltd v. Telco Holding Ltd, etc.*[1274]

First, with regard to the patent law's technicity requirement,[1275] we try to find a solution to a technical problem. To achieve this goal, we only integrate those algorithms in our examination that are actually applicable on a computer, hence defining a general rule for a behaviour that can be executed on and followed by a computer. 697

Second, we only examine qualified algorithms that provide a teaching to solve the named technical problem. We are looking for "man made" rules that represent solutions to complex technical problems.[1276] This teaching, by definition, has to exceed the scope of a scientific truth – such as a mathematical formula – that solely expresses or describes the relationships between mass, force or other natural phenomena that are observable in nature.[1277,1278] It would, for example, not suffice if a formula was presented that, although calculable by a computer, could also be completed by a human without technical assistance.[1279] Similarly, if a formula is the only existing way to express an observed behaviour, it might rather be a description of a natural phenomenon, a discovery, and not an inventive teaching.[1280] This aspect however needs careful evaluation and substantial amount of technical expertise. 698

Further, the teaching that is based on the inventive algorithm needs to be specific and tangible. It cannot be an abstract idea, where no insight into the potential mode of application is provided. Instead, the informed and specialized 699

1273 The following procedure may be used as a simple guideline for the evaluation of an algorithm's patent qualities, but it is by no means exhausitve. As Boecker notes, in practice it remains very difficult to distinguish between common and more sophisticated algorithms (Boecker, 140).

1274 *Aerotel Ltd v. Telco Holding Ltd, etc.*, and *Neal William Macrossan's Application*, decision of the EWCA of October 27, 2006 (EWCA Civ 1371), particularly c. 7 and 40 ff.

1275 See N 291 ff., particularly N 295, N 301, N 303 and N 507 ff.

1276 U.S. Congress (1992), 151; see also discussion in: Laurie, 257.

1277 An example of a scientific truth would be 'F=ma' or 'a2 2ab b2=(a b)2'. These scientific truths are not patentable.

1278 *Gottschalk v. Benson*, 409 U.S. 63 (1972); interpreted in U.S. Congress (1992), 47 ff. and 150 f.; see also N 295, N 301 and N 303 ff.

1279 *Gottschalk v. Benson*, 409 U.S. 63 (1972).

1280 Concluded from Dijkstra, xvii.

reader should be able to obtain the understandable, uncoded instructions on how to execute the teaching.[1281] The examiner should also be able to evaluate the added value the algorithm entails.

700 Finally, the algorithm has to fulfil the requirements of novelty and applicability to be protected under patent law. While the industrial applicability of an algorithm is rarely a problem – if an instructive teaching is provided – novelty and non-obviousness are more difficult to evaluate. An algorithm should not be legally protected if it solely represents the most intuitive way of expressing a particular behaviour in a rule. Software engineers in this case talk of 'the natural one', referring to the most intuitive and natural definition or formulation of what can be observed.[1282] This algorithm is obvious in the eyes of an expert. In order to distinguish specialized from elementary algorithms, Ensthaler and Moellenkamp offer a differentiation by stages into basic algorithms and complex algorithms.[1283] They refer to *basic algorithms* when talking about basic modules that are frequently used in software engineering, such as search or string processing algorithms and mathematical rules.[1284] This type of algorithm is, as its name implies, fundamental and applied in daily engineering work, and should therefore be assigned to the public domain. Algorithms that solely offer an abbreviated or shorter version of a known algorithm should also be excluded from patent protection if they do not offer any additional inventive contribution.[1285] *Complex algorithms*, on the other hand, are algorithms of a second degree, which show a first minimum structure of a potential solution to the technical problem. This means that the functional solution to a problem is already concretized or enhanced to a degree that a tangible teaching is recognizable. This type of algorithm usually combines basic algorithms and "less complex" algorithms as iterations and sequences.[1286] The more refined and enhanced the functional solution to the technical problem is, the more the algorithm should be qualified as complex. Although there are various ways to transform a basic algorithm into a complex one, it is most difficult in this case to overstep the sphere of the mere common. It is only if the engineer leaves

1281 This approach corresponds with the current practice for the evaluation of abstract ideas. For more information, see N 325 ff. and N 326 in particular.

1282 See description in: DIJKSTRA, xvii.

1283 ENSTHALER/MOELLENKAMP, 152 ff. They seem to have based their subdivision partly on SOMMERVILLE, 4th edition of 1992, see particularly vi, 133 f., 169 ff. and 590 ff.

1284 ENSTHALER/MOELLENKAMP, 152 f.; see also FLOYD, 459; STRAUB (2011), N 18.

1285 According to the opinion presented in this thesis, abbreviations of an algorithm constitute a literary rather than inventive achievement, if no further inventive contribution is offered.

1286 ERNSTHALER/MOELLENKAMP, 153 f.

the reliable, established path and finds his or her own particular and successful inventive solution, and expresses it in a formula, that the algorithm is eligible for patent protection.

According to the software companies, inventive algorithms could, for example, be found in the following functionalities:[1287] 701

- providing a faster, accelerated process;
- offering a more efficient procedure;
- achieving an entirely new technical result, such as a process;
- offering a digital solution to a problem that previously could only be solved with an analogue one, and offering further benefits with it;
- achieving a better performance of the device used;
- increasing a process's performance or offering more flexibility;
- increasing the usability;
- offering a new teaching, for example how to steer a particular process.

All these potential applications could represent an innovative use of an algorithm. The decisive factor is the particular development and what it has to offer. If the algorithm passes the test described above, it should be eligible for patent protection. 702

If the algorithm is considered as an idea or rule worthy of patent protection, it is protected independently of its expression or form.[1288] Therefore, it is not only the algorithm's formula that is sheltered but also the behaviour it creates or its expression in a diagram. Also, the source code may qualify as an illustration of the algorithm. The engineer is free as to how he or she wants to illustrate the algorithm, as long as the reproduction still has the exact same function and is still able to fulfil the same process or task. 703

4. (Graphical) User Interface

The user interface is the outermost layer of the computer program and is used to communicate between the software process and the user who wants to steer it. Its overall importance and value in a computer program has increased as the users' expectations of the visual representation of a computer program, and its usability, have increased. As it represents the visual expression of the computer program, the user interface is generally covered under copyright. 704

1287 See results in N 470 ff., N 473 in particular.

1288 See N 266.

705 The user interface reflects the developer's aspiration. As the U.S. Congress has recognized, the developer selects which functions his or her program will perform for the user and what the interaction between the program and the user will look like.[1289] To a large extent, this involves creating an attractive and appealing working environment for the user who buys the program. The visual interface, its individual elements and their composition are selected and positioned consciously. Where the characteristics of a user interface are shaped by the creative and personal decisions of a software developer rather than by the mere functionality of predetermined techniques, copyright protection should be granted. This creativity can for example be integrated into the design of frames and patterns, such as the length and form of the lines, how they are connected or how points are applied. The developer can also play with the position or substitution of objects. Hence, creativity can be integrated into the user interface in very different ways.

706 To evaluate the scope of copyright, it is important to assess to what degree the user interface is a separable expression of a computer program, and to what degree it is predetermined by the functional instructions in the source code:[1290] As Wittmer explains, there are certain visual elements in a layout that mainly constitute a different representation of the same input data processed in the program.[1291] These are therefore functionally conditioned by the instructions in the source code, and are thus path-dependent. Again, as Wittmer puts it, they emerge as illustrative by-products, i.e. based solely on the software engineering task, which is why they cannot be protected.[1292] But the user interface also contains room for creative design decisions. Layout characteristics are used and design decisions are made several times during the development process, starting in the first sketch, to the concept, to the visualized coded and designed layout, up to the final product the client receives. Contrary to Wittmer's view, creative decisions for the user interface do not simply appear at the end, so that only the form-determining elements are worthy of protection, but are made alongside the development process, long before any kind of data is actually processed. The visual elements are added without having a technical function. Instead, they aim to address the user in an appealing,

1289 U.S. Congress (1992), 129.

1290 For more information on functionality, see N 353 and N 364.

1291 Wittmer, 27 ff. and 100; see also U.S. Congress (1992), 126; Commentary to the German UrhG (Loewenheim/Spindler), § 69a N 7.

1292 Wittmer, 112 f.

maybe even artistic manner. But I agree with Wittmer to the extent that visual interfaces should not be protected if they are only a "by-product" of software engineering and do not offer any creative illustrative component.[1293]

At the same time, I do agree that there are parts of the interface design that are merely functional without being by-products of data.[1294] This is particularly true if the design has to be adjusted to certain established user application design standards or best practices that every software product has to offer in order to function as an easily applicable intermediary between the user and the program.[1295] The same applies for certain parts within the usual interface that have to be included in order to trigger the technical processes behind the interface, such as steering buttons. But beyond these predetermined necessities there is a creative potential in the nature of external design that allows the developer to exercise his or her artistic qualities. At the same time, we have to distinguish between ideas for visual interpretations and their actual expression in a visual interface. As the pictorial form of displaying information becomes more prominent with the rise of new technologies,[1296] it is even more important to also guarantee the free use of visual ideas and to ensure that only visual expressions of a certain originality and expressiveness are legally protected. 707

Under certain requirements, user interfaces are also eligible for industrial design rights or trademark protection.[1297] To fall under the design right's scope, these visual elements would have to display a systematic design concept of ornamental and asthetic aspects.[1298] As design rights, due to their exclusionary qualities, might have similar blocking effects to patent law, the governmental institutions responsible for granting design rights should carefully evaluate and test the requirements for their substantive quality. User interfaces are also eligible for trademark protection if they are characteristic of a particular company or service and exhibit enough distinctiveness from other products, for example through implementing a specific corporate identity in a user interface. Applying trademark protection to computer programs is reasonable if the user could potentially be deceived regarding the software's origin by its visual 708

1293 Wittmer, 112 f.

1294 See N 353, N 364 and N 663, regarding predetermined elements.

1295 See discussion in Bullinger/Faehnrich/Ilg, 942 f. and 946 ff.

1296 Fisher, 17.

1297 See above N 215 ff. and N 218 ff.

1298 See Orell Fuessli Commentary to the Swiss DesG, Art. 1 N 32; Staempfli Commentary to the DesG, Art. 1 N 18.

appearance or name. Apart from that particular situation, there are however more accurate options for protecting a technical or creative component than trademarks. The user interface may also be subject to protection against taking undue advantage of somebody else's achievement under unfair competition law. However, in practice, a simple copy of the look-and-feel will rarely be possible technically.

709 The user interface is the visual expression of a computer program that tries to address the users' needs and wishes in using the program. The visual elements in software should be fully eligible for copyright protection if its overall impression is not predetermined by the program's functionality or standardized design criteria.

5. Look-and-Feel

710 The look-and-feel represents a combination of the visual interface and certain functionalities in a computer program. Like the user interface, it is used to communicate with the user. Apart from making the interface neat, it also contains technical aspects that can be separated into predetermined functionalities, and features that are added voluntarily to increase the usability and aesthetics of the look-and-feel.[1299]

711 The look-and-feel can be divided into the navigational features, the layout, the imagery and colours, the interface cue, and the architecture of presented information.[1300] These aspects can be designed to influence the usability, the security, the performance and availability of a system.[1301] It is the developer's task to define what functionalities the program has to meet, and how best to visualize them for the users. As Samuelson et al. and Zelle et al. note, the look-and-feel is rich in creative surface design.[1302] There are a lot of details that have to be considered, such as how the program best obtains the required information from its user, how the user can apply a certain function or feature and how the program behaves in a case of error.[1303] It is in these small details where a lot of the developer's creative leeway can be implemented.[1304] On the other hand, the

1299 See N 147, N 531, N 540 and N 571; see also FISHER, 16.

1300 See Motive Glossary, "Look-and-Feel", available at <http://www.motive.co.nz/glossary/-looknfeel.php> (retrieved August 4, 2019).

1301 BAHIR DAR UNIVERSITY, 13.

1302 SAMUELSON ET AL., 2334; ZELLE/SCHLECHTNER/SCHMID, 9.

1303 CARLETON, 418.

1304 See N 454 ff.

look-and-feel tries to address the needs and preferences of the users as well as possible in order to increase applicability. It consequently requires a lot of vertical domain know-how, in order to best address the user's needs or achieve a specific steering mechanism.[1305] After all, the look-and-feel of a software product is one of its main assets, making the look-and-feel as valuable as the algorithm behind it.[1306]

As the look-and-feel is part of the surface design, it is directly accessible to third parties. While the users are usually able to perceive the visual interface, some of the hidden features are not easy to recognize even though they increase the usability of the program. Still, the trained eye is able to identify these characteristics, making the look-and-feel vulnerable to imitation, which is why legal protection is important.[1307] The look-and-feel can contain both technical innovations and creative aspects, and should consequently be open to both, patent law and copyright protection: 712

According to the view presented here, the look-and-feel is partly open to *patent protection*, if a particular idea that is enclosed contains a repeatable teaching that solves a technical problem.[1308] This may, for example, lie in the combination of certain features that solve a particular technical problem, or an overall concept of features that address the users' needs in an enhanced form and assist the user in performing a technical task.[1309] A simple business method cannot be considered as a patentable part of a look-and-feel, if no continued, repeatable and guided interaction is foreseen and no particular possibility of application is explained.[1310] It is further important to keep in mind that the final layout is often influenced by the concrete design it shows and the functionality the look-and-feel has to achieve. The technical achievements that meet the inventive step can generally be patentable. There are, however, many features that are obligatory in a look-and-feel or have become standard, 713

1305 See N 442 and N 531.

1306 See N 531.

1307 See N 534 and N 540.

1308 For more information on the patentability of functional elements, see N 293 f. and N 298.

1309 See N 303, N 478 and N 531 f. With regard to the decision of the EPO Board of Appeal of January 20, 1985 (EPO T 0605/93) – *Dai Nippon*, and the decision of the EPO Board of Appeal of March 5, 1997 (EPO T 0333/95) – *Interactiveanimation/IBM*, we can assume that visual elements with functionality are open to patent protection; see the same interpretation in European Guidelines for Examination of the European Patent Office, Part G-II-3.6.2; see also recent decision in BGer of July 16, 2020, 4A_609/2019.

1310 In this case, the combination of functions and features represents an abstract idea that is excluded from patentability.

such as classic pull-down menus, frames and windows.[1311] These 'standard' aspects should not be patentable, as they need to be kept free to use for the public. This issue can be illustrated with the help of a car rental look-and-feel, analysed by a report of the Bahir Dar University.[1312] Commonly the online service has to fulfil a large list of functional requirements, such as offering a list of available cars that is always up-to-date, enabling the customer to look at the available cars on the web page and find out what characteristics the cars have, allowing the customer to make an online reservation, and so on. The implementation of these standard functionalities often looks quite similar on the surface. These elements are thus determined by the software product the developer wants to offer, they are common and not novel from a patent perspective, and should thus be excluded from patent protection. Apart from these predetermined aspects, the look-and-feel could be subject to patent protection if the concept, how the users' needs are addressed or their behavior guided by certain features, offers a teaching that solves a technical problem.

714 The look-and-feel also usually contains creative decisions beyond the desired functions. To the extent that there are several ways to implement an idea and it is not predetermined by certain obligatory functional requirements and does not simply represent an expression of codal instructions,[1313] the result represents a creative expression, and thus a copyrightable work. The look-and-feel could thus also be subject to *copyright protection*. This applies in particular to creative interactive forms of presentation, audiovisual creations and strikingly unique structural elements. To assess the copyright quality of a look-and-feel it is important to distinguish the idea and technical implementation of a specific feature from the artistic, facultative manner in which a specific feature is displayed. Due to the high functionality of the look-and-feel, in practice only a few will to reach the expected quality, expressiveness and originality to be protected under copyright. At the same time, the scope of protection of these look-and-feel are likely to be narrow and, in many cases, limited to very similar or identical designs of work. To the extent that the look-and-feel represents a creative expression, it should nevertheless be copyrightable.

715 Finally, a look-and-feel may also be subject to protection against taking undue advantage of somebody else's achievement under unfair competition law. However, in practice, a simple copy of the look-and-feel will rarely be possible technically. In theory, it might also be open to industrial design right protec-

1311 See N 571.

1312 Bahir Dar University, see 11 in particular.

1313 See above N 571; see also discussion in Carleton, 408, 427 ff. and 431; Straub (2001b), 5.

tion. As the look-and-feel is usually based on a systematic design concept, the organization of the implemented features combined with the visual interface may exhibit novelty. However, in practice, only few look-and-feel will provide organizational aesthetics that go beyond mere technical functionalities, that are explicitly excluded from industrial design right. In the case of visual distinctiveness, a look-and-feel may also be eligible for trademark protection, particularly where it is aligned with a software company's corporate identity.

6. Features and Functions

Functionalities determine how software behaves and what it achieves.[1314] Software engineering is still a rather new field of science. For this reason, huge potential lies in the software field to invent new teachings and technical solutions.[1315] According to the findings in the interview study, new inventions in the ICT field often involve new functions or features. Examples of such inventive components are push messages, guiding mechanisms such as Google maps, and new database environments or business models that introduce new selling mechanisms. But the digitalization of established analogue processes can also represent a new key feature, if an inventive teaching is included. All these achievements tend to involve an increase in the usability of a software product, optimizing its performance or improving the security of a system. 716

It was mentioned in the interview series that functions and features require a lot of domain know-how and experience to make them. It is only if they are developed logically and are structurally thought out that they can fulfil their purpose. Their development also requires a lot of resources, as a lot of testing, simulation and brainwork is needed to achieve a satisfactory solution. They hence represent a very valuable part of the computer program.[1316] At the same time, the functions and features of a program are commonly exhibited on the surface of a computer program. As the trained eye can recognize them easily,[1317] they therefore require legal protection to secure them. As the functions and features merely show a technical idea or character rather than a creative expression, they are generally excluded from copyright protection. 717

The functions and features may be subject to patent law protection if they offer a unique technical solution that involves a specific way of application. This 718

1314 SAMUELSON ET AL., 2316, and above N 148.

1315 See N 470 ff.

1316 See N 532.

1317 See N 540.

is also true for functions or features that technically represent a business method. To test their eligibility for patent protection, functions and features have to fulfil the same requirements as an algorithm would have to.[1318] First, for obvious reasons, only functions and features that can be implemented and applied in a computer program are included in the evaluation. Second, only qualified functions and features that offer a "man made" teaching to solve a technical problem, and exceed the scope of a scientific discovery, are allowed. Functions and features that merely describe events that are predetermined or naturally caused by the setting are excluded from the patent scope. Third, the presented teaching, including its added value needs to be specific and tangible, and not represent an abstract idea. Otherwise, it would be excluded from patent law protection because it represents unprotectable business methods. If an informed and specialized expert would understand how he or she had to apply the teaching, the functions and features can be protected with patent law. Finally, the function or feature has to fulfil the requirements of novelty and applicability to be protected under patent law. Neither a function nor a feature can be legally sheltered if it just provides the most intuitive way of expressing a particular behaviour in a descriptive rule. The non-obviousness criterion is particularly important to assess functionalities that are based on business methods. Only objects with a minimum inventive threshold fall into this scope. Otherwise, the public's interest in the unrestricted use of common functions should be prioritized.[1319] If a unique set of functions or features fulfils these criteria it is eligible for patent protection.

7. Development Documentation

719 The development documentation commonly contains a lot of the creative thinking that is important for the final implementation of a software product. This creative thinking is expressed with the help of sketches, graphs, keywords and sometimes also early-formulated codes or algorithms. It retains the temporary working products in different stages alongside the development process.[1320]

720 It was outlined above that the copyrightability of the development documentation, such as sketches and feature concepts or first versions of algorithms, is

1318 See above N 696 ff.; see also description and explanation in European Guidelines for Examination of the European Patent Office, Part G-II-3.5.1.

1319 For this, the same can be repeated as outlined for certain algorithms that need to be kept free: See N 696 ff.

1320 See N 144 and N 166.

still unsettled.[1321] One of the most sophisticated critics is Wittmer. He stresses that this form of documentary material is not necessary for the development of a computer program but only helps to maintain it. The documentation would only be used for a limited short time and then would be replaced with the final product. He concludes that these temporary concepts are not valuable and therefore should not be protected by copyright. Only conceptual documents that are then passed on to clients, such as input–output models or detailed design plans – and are hereby shared with third parties – should be legally protected.[1322] I believe that the described scenario no longer applies for the practices in software development. Kummer, back in 1968, stated (in a differenct context) that copyright law would not ask whether a piece of work was in a mere preliminary stage, represented only a first idea, or parts of a planned or begun work, but instead would only consider the individual distinctive fragment and evaluate whether an original work existed.[1323] It was outlined in the interview series that the development documentation represents a creative expression of visual or literary information.[1324] If this information falls into the hands of third parties, all the technical and design solutions that were elaborated at the expense of the developer's time resources are freely exposed. The personal domain know-how as well as the developer's creative application of personal expertise and experience is directly accessible. It also emerged in the interview series that it is the development documentation that is particularly exposed prior to the closing of a contract in commissioned work, where the concepts and sophisticated drafts are presented to potential customers without contractual protection.[1325] The companies made it clear that if sensitive products, such as the development documentation, were not covered by IP protection, they would be forced to keep them for themselves instead of opening them up for clients or the general public. This consequence would not only limit free know-how transfer but would also prevent the right holders from trading their goods effectively. At the same time, in cases where they were obliged to grant insight into their preliminary products, the risk of infringement would increase significantly. Companies who pitch for contract work are thus exposed to an extended, avoidable risk.

The development documentation can contain the required creative quality in the form of a contribution to fulfil the copyright criteria and fall under its 721

1321 See N 356 bullet point 10.

1322 Wittmer, 34 f.

1323 Kummer, 77.

1324 See N 541, N 544 and N 549.

1325 See N 500 ff.; see also discussion of (prototype) contests in Scotchmer (2006), 47 ff.

scope of protection. At the same time, it also earns legal protection with regard to the resources required, and the knowledge exposed in the documentation. However, the quality of development documentation can vary considerably, which is reflected in its protectability through copyright. The international copyright community expressed in the wording of Art. 2 para. 1 RBC that drafts, in general, are eligible for copyright protection, if the requirements are met. Still, to my knowledge, there has not yet been a case where IP rights on the development documentation could be enforced. The main reason for this may lie in the fact that it is often difficult to delimit simple notions from elaborated visual and literary concepts in development documentation. In order to fall under the copyright scope, the presented material should surpass the level of an abstract idea, and provide a recognizable and original possibility for implementation in an expression.[1326] It must be presented in a way that allows analysis of whether it fulfils the copyright requirements.[1327] To do so, the functional elements must be sorted out. Both require a legal assessment as to when the result of the work appears sufficiently concrete. If this creative step is accomplished, the development documentation should be eligible for copyright protection.

722 As well as copyright protection, the development documentation may also fall under the scope of work product protection and secrecy under unfair competition law if the information disclosed builds on relevant business or technical information that guides the manufacturing or commercialization process.[1328] In practice, however, the protection of secrets usually fails due to the strict legal requirements and a rather complex court procedure. For this reason, the focus instead lies on contractual non-disclosure agreements.[1329] According to the interviewed companies, in the software industry parties tend to refrain from signing confidentiality agreements prior to a formal commission contract because potential principals often do not want to commit at this early stage of contract negotiations and unnecessarily restrict their room for decision-making and in selecting a supplier.[1330] An automatic and expressly statutory legal protection mechanism that does not require any formalities, such as copyright, would thus be better able to meet the basic needs and practicalities of

1326 See above for the legal background to the exclusion of ideas in copyright in N 350 f., the interviewees' suggestions in N 548, and below regarding delimiting abstract ideas from concepts in N 725 ff.

1327 See also THOUVENIN/BERGER, 6/3.5., 2.

1328 See above N 223 ff.

1329 See N 229 ff.

1330 See N 500 ff.

the software companies.[1331] As the development sketches are usually further refined before they reach their final version in the software product, they often do not fulfil the requirements for protection against taking undue advantage of somebody else's achievement under unfair competition law.

8. Tabular Summary

To summarize the above, the following table gives a short overview of the legally relevant software elements and the possibilities for protecting them 723

Software component	Legal Institute
Whole Software Product	Copyright protection. Excursus: protection against taking undue advantage of somebody else's achievement under unfair competition law.
Code	Copyright protection of the literary formulation of the source code as well as its organization in structures and sequences; Copyright protection of the machine code. Excursus: protection against taking undue advantage of somebody else's achievement under unfair competition law.
Algorithm	Patent protection.
(Graphical) User Interface	Copyright protection; Excursus: industrial design right protection for certain visual elements or trademark protection for the overall distinctive visual apperance. Within a limited scope of one-to-one copying also protection against taking undue advantage of somebody else's achievement under unfair competition law.
Look-and-Feel	Patent protection for the combination of certain features to meet a particular functional problem, or an overall concept of features that addresses the users' needs in an enhanced form; Copyright protection for audio-visual elements in a look-and-feel that are not predetermined by functional elements; Excursus: industrial design right protection for certain visual elements or trademark protection for the overall distinctive visual apperance. Within a limited scope of one-to-one copying also protection against taking undue advantage of somebody else's achievement under unfair competition law.
Features and Functions	Patent protection.

[1331] For larger, more cost-intensive projects, a deposit with an escrow agent is often used in this case. However, this is less common when pitching with small and medium-sized companies.

Development Documentation	Copyright protection; Excursus: work product protection and secrecy under unfair competition law.

D. Not Protected Elements in Particular

724 Having established which elements may be covered under the copyright and patent subject matter and scope, we now turn to elements which are not covered. The next section discusses two particular types of elements that occur in developments and may generally be considered as not legally protectable. First, copyright does not protect ideas. For this reason, the raw idea has to be delimited from its creative expression in a computer program. As this question only applies to copyright, this first section focuses on this field alone. Second, there are certain elements in developments that may be considered as either common in the specific context or determined by the particular setting of a development. With regard to standards, necessities and best practice, this second section discusses to what extent these elements occur and how they affect the legal protection of a good. The aim here is to discuss how to apply these legal exceptions to computer developments and what particularities to keep in mind while doing so. The conclusions of the first two sections are then used to build a possible testing approach in a third section, the so-called black box test.

1. Distinguishing between Ideas and Expressions

725 Copyright law builds on the principle that ideas are free to use.[1332] While patent law protects every potential realization of a specified invention, independent of its implementation, in copyright only the expression of an idea fixed in a particular creative work is eligible for protection. For this reason, the author in copyright has to prove, if contested, that his or her work has overcome the step from an idea to a concrete realization and that it is affixed in a specific

[1332] Art. 9 para. 2 TRIPS Agreement; see N 350 f.

expression. How the individual elements were realized in a final expression is essential in copyright.[1333] This section discusses, from the perspective of copyright, how ideas can be demarcated from expressions for computer programs.

726 In order to distinguish between ideas and expressions, U.S. copyright law currently applies the so-called *merger doctrine*. It says that, if there is only one way to express an idea in a piece of work, its embodiment does not represent a copyrightable expression of the idea but rather an uncopyrightable form of the idea itself. However, if an idea can be implemented in a work in several different ways, it may be considered as a potentially copyrightable expression.[1334] The courts emphasized in *Apple Computer, Inc. v. Formula International, Inc*[1335] that the purpose or function of a utilitarian work represented the work's idea, and everything that was not necessary to achieve and illustrate that purpose or function would be part of the expression of this idea. *In Whelan v. Jaslow*,[1336] the court supplemented: "Where there are various means of achieving the desired purpose, then the particular means chosen is not necessary to the purpose; hence, there is expression, not idea."

727 Despite how valuable the merger doctrine may sound, it remains a merely theoretical model that is difficult to implement in practice. It becomes better applicable when combined with the *abstraction test*.[1337] In *Nichols v. Universal Pictures*[1338] the court provided the following guidance with regard to the applied step-wise abstraction: "Upon any work, and especially upon a play, a great number of patterns of increasing generality will fit equally well, as more and more of the incident is left out. The last may perhaps be no more than the most general statement of what the play is about, and at times might consist

1333 An applicant in patent law has to define his or her idea and show how it is used and implemented in practice so the idea is considered specific enough. The applicant does not need to name or circumscribe one or all potential applications of the invention, if the idea was described in a detailed manner so that the instruction could be applied to solve the particular problem. How abstract ideas can be distinguished from inventions in patent law is discussed in the legal foundation in N 350 f. Due to the rich U.S. case law, such as *Digitech Image Techs., LLC v. Electronics. for Imaging, Inc.*, 758 F.3d 1344 (Fed. Cir. 2014), there are clear criteria as to how this demarcation has to be done. The following section will therefore not cover this question.

1334 For more information, see above N 351 f.

1335 *Apple Computer, Inc. v. Formula International*, Inc., 562 F. Supp. 775 (9th Cir. 1983).

1336 *Whelan Associates v. Jaslow Dental Laboratory*, 797 F.2d 1222 (3d Cir. 1986).

1337 See also above N 351 f.

1338 *Nichols v. Universal Pictures Corporation*, 45 F. 2d 119 (2d Cir. 1930); see similarly in Haberstumpf (1993), II N 60 f.

only of its title; (...) there is a point in this series of abstractions where they are no longer protected, since otherwise the playwright could prevent the use of his ideas, to which, apart from their expression, his property is never extended."[1339]

728 At first, this explanation sounds very conceptional. But its statement is decisive in understanding what may be considered as an expression and what is not. What it means is that the more general and generic an implementation of an idea is, the more we should come to the conclusion that the implementation is not a real expression but rather only the basic concept or abstract idea itself.

729 This can be illustrated with a simple example we all know from our childhood: the story of a woman being saved by a noble knight. This basic story concept is widespread in literature and has found its way into many different storylines. The more elements are added to the basic concept, the more colourful the story becomes and the more the original idea fades compared with the expressed version. In this way, the original idea of a knight saving his woman can result in, for example, the Greek mythology *Eurydike*, Grimm's fairytale *Snow White* or Walt Disney's *Frozen*. The abstract idea included in all three examples is embellished with its own individualized peculiarities.

730 The abstraction test can also be applied to computer programs. In *Lotus v. Borland*[1340] the court had to evaluate whether a particular hierarchy of a user interface's top menu structure should be considered as copyrightable. Top menus build on the idea that all the important information and directions are visible on the start screen at one glance. The court ruled that a simple 1-2-3 menu hierarchy generally does not represent a creative expression if the implementation of this basic idea of a menu's structure is limited to stringing together a number of icons on one screen in a numerically structural way. It declared that in the present case the abstract idea and its expression were considered as merged and, thus, the result was not copyrightable. In order to be copyrightable, the menu structure would have needed to add further elements so that the idea became more ornamented and original. It has to outline additional creativity. Although the court tried to circumscribe the idea term and determine a way that ideas could be distinguished from expressions with the abstraction test, the court failed to define the problem of a program's

1339 *Nichols v. Universal Pictures Corporation*, 45 F. 2d 119 (2d Cir. 1930), 121.

1340 *Lotus v. Borland* 516 U.S. 233 (1996).

menu hierarchy where it lay: that the utilitarian idea of the program's menu and its implementation with a 1-2-3 menu hierarchy were too close and conceptual and, therefore, merged.

731 Although the merger doctrine was made better applicable in court with the U.S. abstraction test, it remains a highly sophisticated model. As Ogilvie explains, applying the test in practice requires highly specialized knowledge of the involved technology, processes and concepts.[1341] Also Nimmer et al. remark critically that the test is not able to provide a straight answer of "where the dividing line exists for a given work, but rather [it] provides a method of analyzing a work to determine where the line should be drawn".[1342] Or as Lemley noted, getting to the heart of it: "There remains a good deal of misunderstanding about what exactly it means to 'abstract' and 'filter' (...)".[1343] Although I completely agree with the critical conclusions of these authors, I also believe that the U.S. abstraction-and-filter method remains the best theoretical model currently available in law to distinguish ideas from expressions. It may also serve in European and especially Swiss copyright.

732 The question remains where the border may be set between an abstract idea and its practical implementation and where the necessary originality threshold, the so-called "Schöpfungshöhe" or creative step is overcome. In the interview study, the software companies described that the final outcome needs to exhibit a "minimum level of maturity" or a "minimum level of substance". They explained that an idea has to be elaborated to the extent that it leaves the sphere of being abstract.[1344] In order to achieve this level, the idea has to be fixed in a particular context and, thus, become seizable. In intellectual property law – and copyright and patent law in particular – we generally look for creative room where the creator makes certain decisions and thereby offers his or her *personal contribution* to the idea. The difference between the raw idea and its expression therefore can be found in the rich amount of substantial detail that helps to implement the basic idea in a practical way. But it also lies in the outcome of a creator carefully thinking through the creative process and determining the goods' exact final shape.[1345] The creator thus offers his or her personal contribution through his/her own considerations and personal notes. The decisive factor should therefore be whether the expression adds

1341 Ogilvie, 529.

1342 Nimmer/Bernacchi/Frischling, 636, with further references.

1343 Lemley (1995), 3.

1344 See N 547 ff.

1345 See also above N 454 ff.

considerable and substantial details to the raw idea. If the result exhibits additional creative elements that are not solely used to fulfil the purpose of the idea but, instead, make the basic idea become ornamented and original, the expression can be filtered from its notion in the abstract idea. What is the exact plot of the knight's rescue story? How exactly does the knight rescue the noblewoman? And analogously, how is the screen structure of a computer user interface built in detail? Does a particular colour pattern, animation, positioning of the buttons, sounds or 2D-organization make the expression special so that it is not a simple implementation of an abstract menu but rather a creative expression? The key question is how the creator implements the individual idea and creatively organizes it within a concrete framework, including his or her personal substantial particularities. *Additional elements* and *unexpected twists* may be added to give an idea its particular drift. What if the noblewoman suddenly rescues the knight, or rather than a knight, it is a normal 'commoner' who rescues the noblewoman? A menu structure could, for example, offer an additional feature that supports the user in a particular way, such as with light effects or animations. If an idea is not implemented one-to-one in an expression but rather contains details or additional particularities, the concept may be raised from an abstract idea into a noticeable original expression of the latter.

2. Standards, Necessities and Best Practices

733 Copyrightable elements are those that represent an original creative expression that do not follow a purely utilitarian or functional purpose.[1346] In order to be considered original, expressions have to exceed what is predetermined, expected or ordinary for the particular category of work in question. Similarly, in patent law, elements that are common and established in prior art are not considered novel.[1347] Developments that include established elements do not offer an improvement on prior art, and thus cannot be protected. In practice, computer developments often build on elements that are either common and utilitarian in the specific context or predetermined by the particular technical setting of a development. The first group involves elements that are frequently used to solve a technical problem or realize a creative idea. The second group includes all elements that do not result from creative or inventive thinking of the creator but instead are dictated by the exact context so that the development can fulfil a particular functionality. The following section discusses how

1346 See N 361 ff.

1347 See N 311 ff.

such common or predetermined elements are used in a software product and what this means for the legal protection of a component or software product overall.

Copyright explicitly excludes functional elements from its protection scope; these elements are considered uncreative.[1348] Contrary to copyright, patent law does not exclude functional elements as such. However, certain technical elements may be qualified as 'given by nature' where an inventor does not 'invent' these solutions to a technical problem but instead discovers a logical effect in the nature of a specific context. As the inventor does not offer his or her own teaching, such 'discovered' elements are considered non-technical and thus unpatentable.[1349] On the other hand, there are certain solutions that are obvious or unoriginal in the specific setting; they represent the most natural solution to a problem or simply the most elementary way of expressing it.[1350] A person skilled in the particular art or science would expect their occurrence. If functional elements fall under the given descriptions, these elements are also excluded from patent protection.[1351] In computer programs functional elements are manifested through *technical necessities*. These elements are required so that the program is able to technically function in its system environment. The particular way a developer has to select and realize their idea may predetermine certain functional or technical elements. These elements need to be integrated for the development to fulfil its goal.[1352] A common example is when an engineer has to work with the programming languages that Apple and Google offer in order to sell his or her product on the main two mobile operating systems iOS and Android. As the functionality dictates these mandatory parts, these elements – here the selection of the programming language and the incorporation of certain code elements – do not originate from the author's creativity or ingenuity but rather from technical necessities. They are therefore considered uncreative and uninventive. 734

At the same time, the interview series revealed that there are other *predetermined factors*, or *necessities*, in a computer program that may be regarded as requisite from a project development point of view. Although these elements are usually not "functional" in the sense that they serve a compelling purpose, they are at least common in developments, they are utilitarian. As Davidson 735

1348 See N 353 ff.

1349 See N 293.

1350 See N 571; see also step no. 4 in the test to verify algorithms in N 696 ff.

1351 See N 316 ff. and N 681 ff.

1352 See N 571; see also discussion in LEMLEY ET AL., 47 f.

notes, these elements are "central" to every software product.[1353] If they were protected under an IP monopoly-like, exclusionary right, others would be prevented from using them and could either not meet the technical prerequisites or not fulfil the users' expectations of a customary software product. According to the interview studies, there are certain engineering standards and expected best practices, for example with regard to user designs that have to be integrated into a program, otherwise today's expected usability of a product could not be served.[1354] Such expected elements particularly include the following:

736 *Engineering standards*[1355] refer to technical processes and product solutions that are not predetermined by the technical environment but are common in computer programs as they are either effective, simple or for some other reason well supported by the software community.[1356] Standardization processes build on the notion that the whole industry tries to agree on one common procedure. In practice, standards are often set by a standard-setting organization in a formal process of an organizational body, such as the European Commitee for Standardization's "Principles and Guidance for Licensing Standard Essential Patents in 5G and the Internet of Things (IoT), including the Industrial Internet" or the European Telecommunications Standards Institute's "Long Term Evolution Standards". But standards may also evolve for practical reasons and are thus called de facto standards. As Nimmer et al. explain it, "some methods of wording or searching are significantly more efficient than others in handling particular types of data, even though any of numerous methods will work. (…) Where the efficiency trade-offs between methods are substantial, as is often the case, common sense would dictate that a programmer chooses the most efficient method. (…) The availability [of other methods] is theoretical".[1357] The software developer has (almost) no other choice but to make use of a particular element. Replacing or reformulating it would be unreasonable. In this case, a product or process prevails because market participants consider it as the best or do not see an alternative possibility to solve a problem.[1358] A classic example of this is security certificates in the form of certain encryption technologies.

1353 Davidson, 1080 f.

1354 See N 571; same result in: decision of the ECJ of May 2, 2012, C 406/10, *SAS Institute, Inc. v. World Programming, Ltd.*, c. 40.

1355 For more information on standards, see also N 845.

1356 See N 398 and N 571; see also Sommerville, 706 ff.

1357 Nimmer/Bernacchi/Frischling, 641 f.

1358 Hilty/Slowinski, 781; Lemley et al., 44 f. and 47 f.

Best practices, on the other hand, represent elements that are integrated in software products because the users or customers expect them, because they are common in a computer program.[1359] Lemley et al. refer to these elements as "elements dictated by external factors".[1360] An example of such an element is the shopping cart in online stores or the printer icon in a document-editing program. If these elements were missing from the user interface or look-and-feel, the customers' user experience of the software product would most probably decrease, affecting their desire to use or purchase the software. In contrast to functional elements and engineering standards, best practices are not predetermined by technical settings, or a matter of efficiency or compliance. Instead, a software developer has to integrate them in his or her computer program because they are socially established and expected. 737

In both cases – engineering standards and best practices – the author or inventor is usually limited in his or her creative process. They have to integrate these elements because they are expected and common in a software product. If this is the case, the creator most often cannot offer any personal contribution to what was before. Instead, they make use of what was already established. As these elements cannot be classed as original or novel, from a legal perspective, they do not represent a creative or inventive step and are thus excluded from IP protection. Only if additional originality is exhibited, and the developer finds a way to express creative leeway and is able to distinguish his or her expression from the predetermined structural elements, can it be deemed a copyrightable work. 738

How can we determine what is technically or socially predetermined in a computer development or what is considered a functional necessity? The U.S. *scènes à faire* doctrine seeks to determine and exclude elements that are obligatory or necessary to fulfil a certain creative expression.[1361] The doctrine was established for movie settings in *Williams v. Crichton*[1362] and was then adopted to evaluate computer programs in *Altai*[1363]. It says that *whenever the possible way to express an idea is restricted by external factors, the outcoming element is most probably uncopyrightable.* This U.S. doctrine may be equally applied to 739

1359 See N 398 and N 571.

1360 LEMLEY ET AL., 47 f.

1361 For more information, see above N 355.

1362 *Williams v. Crichton*, 84 F.3d 581, 583 (2d Cir. 1996), commenting on *Walker v. Time Life Films, Inc.*, 784 F.2d 44, 52 (2d Cir.).

1363 *Computer Associates Int., Inc. v. Altai Inc.*, 982 F.2d 693 (2d Cir. 1992).

European and Swiss copyright law. In the *Altai* case, the court identified a number of elements that they classified as typically non-copyrightable, such as:[1364]

1. mechanical specifications of a computer on which the programs run;
2. compatibility requirements of other programs the program has to work with;
3. computer manufacturer's design standards;
4. the target industry's demands; and
5. widely accepted programming practices within the computer industry.

740 *The first two elements* may be classed under what was previously referred to as technical necessities; program specifications, interoperability requirements or otherwise demanded functionalities represent a group of elements an engineer has to embed in their program as otherwise the program would not be able to collaborate with or function within the technical environment it is connected to and surrounded by.[1365] These elements are regarded as functional and are thus excluded from copyright protection. However, from a patent law perspective, these elements may also be regarded as technically predetermined; if a developer solves a technical problem with their help, a technically skilled person might consider them to be obvious, in that they are a logical consequence of the specific technical setting. *The third point* represents a sub-group of technical necessities, such as Android and its programming language APK, which have to be implemented in order to publish compatible apps on the platform.[1366] *The fourth point* the court indicated is a very vague one and requires further explanation to understand its range. It may refer to what the users expect from a computer program in terms of best practices, such as a neat structure, a logic order or common processes. These elements, again, are widespread in software programs and may thus be considered as established and common.[1367] However, the target industry's demands alone do not preclude IP protection. Realizing a market need in a creative or inventive way is one of the main goals of intellectual property protection in computer programming. If the realization exhibits originality, it may qualify as a copyrightable expression. It is thus not fully clear what the court was referring to in this point. Its formulation is unfortunate. *The final point* the court outlined

1364 *Computer Associates Int., Inc. v. Altai Inc.*, 982 F.2d 693 (2d Cir. 1992), c. 2b.

1365 See N 734.

1366 See N 734.

1367 See N 737.

refers to the classic engineering standards which are used regularly in software engineering.[1368] These are widely established and represent significant basic elements in software engineering, such as lockout codes. The influence of the Open Source community and established libraries may have a similar effect. As these engineering practices are widely used in computer engineering, they usually are not able to fulfil the originality criterion in copyright, or novelty in patent law, and therefore do not represent a creative or inventive step. The elements the court classed as not eligible for copyright correspond with the findings of the interview series. They can all be put into the categories outlined in this thesis of technical necessities, engineering standards and best practices.

741 While some elements in a computer program can be quite easily identified as predetermined factors, others require more fundamental technical knowledge to evaluate them correctly. In *Lexmark Int., Inc. v. Static Control Components, Inc.*,[1369] the court assessed whether a printer toner driver to a specific printer was copyrightable. They focused on the question of whether the technical access to the toner through authentication was possible by expressing the access function in another source code. In their understanding, if there were different ways to express the same function, the source code was not functional but instead open to copyright protection, which is what they concluded. What the court overlooked, however, is that although there might be a different programming language or another phrase to express the same function, manufacturers are usually able to create a particular system environment where the machine is only open to certain programming languages, carefully selected to exclude third-party providers offering interoperable substitute devices. The court failed to look into possible engineering standards that preclude certain technical approaches. It may be argued therefore that the court did not take into account that certain elements in a computer program are determined by the program purpose itself. This perspective might have changed the result in *Lexmark*. In order to create adequate IP protection it is therefore important to take into account that there are certain elements necessary for a program to work properly. These elements are predetermined and have to be integrated, regardless of whether or not in theory there would be an alternative way to realize this development idea.

1368 See N 736.

1369 *Lexmark International, Inc. v. Static Control Components, Inc.*, 387 F.3d 522 (6th Cir. 2004).

3. The Blackbox Test to Identify Unprotectable Elements

742 The previous two sections have discussed which elements in a computer program are generally excluded from copyright and patent protection due to their properties, and how they may be legally approached with the merger doctrine, the abstraction test and the scènes-à-faire doctrine. The difficulty with these legal models is that whether or not a creative good is eligible for IP protection can often only be determined retrospectively. During the development of the software, developers, project managers and legal consultants do not have a method available to evaluate whether or not the development is eligible for copyright and patent protection. The following approach – constructed as a blackbox test – can support the developers and legal consultants in identifying non-protectable elements in computer programs. It represents a short rule of thumb based on the existing doctrines and supplemented with what was learned from the interview series. However, the approach presented here can only provide a rough description, without making any claim to be exhaustive.

743 A *blackbox* in general is a theoretical model, describing something whose content is hidden, unknown or not entirely perceivable from the outside.[1370] Davidson adopted the concept of a blackbox for computer programs, aspiring to deduce with a legally legitimate method what a program visible or perceptible from the outside may contain in its hidden spheres and content. He thereby tried to delimit eligible from non-permitted third-party activities on copyrighted goods, integrating typical cases such as reverse engineering and potential derivatives.[1371] In his paper, Davidson explains the theoretical background to his model and gives some examples of what his test may achieve. Unfortunately, the paper does not provide any instructions that guide the examiner in applying his test. And although Davidson is successful in concretizing the abstraction test in a different application (putting everything in a box and seeing what is visible from the outside), his test is not much easier to apply than the abstraction test itself.

744 Therefore, I have adapted Davidson's idea of a blackbox test for software to differentiate between elements in computer programs that can be protected, and those that cannot be protected with copyright and patent law. Instead of using the test to discover which components are incorporated in software (within the box) and how they could be applied, I want to use the blackbox test to determine *what we can expect from a particular computer program and*

1370 ISO/IEC/IEEE 24765:2017-3.390.

1371 See DAVIDSON, 1080 ff.

which elements it has to provide to fulfil the technical and social expectations. I am looking to delimit functional and predetermined factors from creative, inventive, original and non-obvious ones that constitute the developer's considerable and substantial personal contribution in a creative step. The present blackbox test also aims to determine to what extent the final product is distinguishable from its abstract development idea and whether a protectable expression is involved. I recommend that it is only conducted by an experienced examiner who has fundamental knowledge of the substance in question and who is trained in conducting such examinations. If several experts were to carry out the test, an even more meaningful result could be achieved with increased objectivity.

The goal is to find those elements in the software product that may be expected because they have to be integrated in the software for it to fulfil its purpose. The idea of the model is that the examiner writes down his or her personal inherent logic. In theory, the examiner puts the whole software project into an imaginary black box. Within the box all the 'mysterious magic' happens that helps to create the software product. It is a creative process that is open to every possible result. From outside the box, we can later only view the elements that can be characterized as typical or necessary in a software product. Throughout the process, these elements become accessible and visible, and are thus no longer hidden within the box. Unlike Davidson's blackbox test, in this test the predictable elements outside the box are not eligible for copyright and patent law protection, while those inside the box involve everything that is more than ordinary, and contains a creative step. The content inside the box therefore is potentially copyrightable or patentable. To get there, the examiner has to decide what is predictable and expected from a certain program that has to fulfil a specific task. It is important that they only list what they are convinced should be included in the program from a technical or utilitarian perspective – the predetermined elements. The examiner should not include elements that he or she thinks it would be good to have. They may solely include aspects they know about or have read about in specialized literature, but not aspects that are guesswork or interpretation. In an ideal scenario, the examiner would organize their notes by separating them for each component, distinguishing between things that are obvious in the technical implementation due to the abstract purpose followed, and elements that represent an original creative expression of the purpose. The process could proceed as follows: 745

746 It starts with forgetting everything the examiner knows about the software product, as it is theoretically in the box. The only thing known is what purpose the software should fulfil.

747 Similar to the *problem-and-solution* test[1372] that the European Patent Office applies to verify a computer program's eligibility for patent protection, we first want to establish the closest prior art for the specific technical problem. For this purpose, the experienced examiner writes down methods that could be used to make the software serve its specific purpose, outlining the software requirements. They will include some established functions they know and other established engineering practices within the industry. They may also do some limited short research as to what the software community recommends for such situations and which programming standards are easily accessible. The examiner does not have to draw up a perfect draft, but just think through the concept and get an idea of the matter in question.

748 It should also be taken into account what the users would expect from a product to make them buy or apply it. This will include features and functions that are widely established on the market. The examiners may review which products they use at home or at work that fulfil a similar purpose to get an idea of what the user experience and surroundings should look and feel like to make the product infinitely substitutable and competitive with similar ones on the market. The examiner only notes the essential elements.

749 Next, the examiner tries to establish which functional elements are required for the software to work within its implementation environment. This includes everything the program requires in order to work properly in its technical surroundings, including the pre-existing system environment and technical specifications.[1373]

750 Once the predetermined functional elements are established, it is possible to work out which programming language should be used. In every programming language, there are some commands that are commonly utilized, either because they are standard or because they are reasonable. The examiner notes all the standard commands they can think of and checks in the appropriate libraries and online forums which ones should be used.[1374]

1372 For further information, see N 319.

1373 See N 735.

1374 See N 736 and N 737.

In a final step, the examiner considers what would be necessary to display the program superficially so that the user can apply it. 751

Having thought through the project, the examiner puts their notes to one side, "opens" the imaginary blackbox and has a look at the actual product, going through the different components one by one. What kind of functions and features does the product build on? Which commands does it express in which programming languages? How is the source code structured? How is the look-and-feel and the graphic user interface constructed or designed? The examiner will then note what catches their eye in the product. He or she goes through the individual elements, not thinking about their personal list of expectations. This might take a while, but is important to fully capture the software. 752

The examiner makes a first selection of which elements they see in the software that are obviously used solely to fulfil the purpose of the software, such as structuring the user interface, and to what extent there are additional elements that substantially ornament the concept of these abstract goals. The examiner goes through each component of the software product and crosses off their list using the notes from the actual software product all the elements that they believe follow the abstract concept of an element without adding something creative to it. The idea and its expression have merged in these elements and they are not copyrightable. 753

Once finished, the examiner takes both sets of notes – the one with their expectations and the one with their observations and goes through each component comparing what they have written down. The examiner crosses off which elements they noted on both of them. If they had been able to anticipate these elements in their list of expectations, those elements have to be rated as either functional or unoriginal. 754

In a second closer analysis, the examiner verifies to what extent their expectations corresponded in the observed software product in order to identify less evident predetermined elements. To this end, the insights of the U.S. Patent and Trademark Office's *functional approach*, are used to evaluate the novelty criterion in patenting.[1375] The examiner first decides if the observed element represents a *direct substitute* of the one they had expected. From a copyright perspective, this implies that the specific element was expressed differently in the observed expression from what was expected, but it still has 755

[1375] The functional approach was first established in *Graham v. John Deere Co.*, 383 U.S. 1 (1966). For further information, see N 321.

to be considered as equivalent because it does not stand out from the expected element, if, for example, the user interface follows the same logical structure, but is organized using alphabetical characters instead of numbers. From a patent law perspective, the direct substitution includes the same behaviour that was described with a different rule, obtaining the same result. A simple example is the first binomial formula, where (a b)2 is the same as a2 2ab b2. If the observed element does not represent a direct substitute of the expected element, the examiner has to look for *equivalent solutions that are obvious to try*. The U.S. functional approach names several concepts that may be considered as 'obvious to try'. For the goal of the blackbox test – to determine the predetermined elements in a software product – I want to limit the variations of equivalent solutions to two applications: First, variations where the observed creation combines the anticipated element with another known element. For example, when the user interface follows the same logical structure as anticipated, but instead is organized with the help of bullets rather than numbers. Second, variations where the anticipated element is combined with an unknown element to arrive at the observed creation, but using a method, suggestion or teaching that is established in the prior art. For example, if the creation uses the expectation of a user interface of being logically structured, but instead of arranging the information by numbers, the information is organized alphabetically or separated with commas instead of divided into sections. All elements that represent such a straightforward substitute or combination of expected elements also have to be regarded as predetermined. Everything that goes beyond the scope of the examiner's expectations has to be considered as independently contributed, and thus potentially creative or inventive.

756 What then remains is a shorter list of elements that are neither obvious according to the purpose the software seeks to pursue, nor clearly necessary from a technical point of view, nor predetermined by standards or best practices. The remaining points on the list can now be individually analysed as to what degree they offer a substantial creative or inventive contribution. They can be tested to see whether they fulfil the copyright requirements of originality and intellectual creation, and the patent requirements of novelty and industrial applicability.

757 The roughly circumscribed blackbox test can structure the previous evaluation and help to identify non-protectable elements in the software product. At the same time, it can help to narrow down the potentially copyrightable products to a lower number of possible outcomes, which can then be evaluated one by one.

E. How to Integrate Newer Development Trends

One of the main findings of the interviews was that software development – although established among scientists and engineers for almost twenty years – is still a very dynamic discipline where new discoveries and practices are continually being made. Although a lot of knowledge and know-how has been consolidated, there is still plenty of room to develop new technical trends and project management methods. At the same time, we face an internationally harmonized copyright and patent law that was created in the late 19th century and since then had only been revised in parts until 2000.[1376] Copyright in particular emanates from the idea that once a creation is fixed in a medium – such as a book or painting – it is not going to be altered any more. While patent law does not consider how an invention is implemented, copyright is rather static and resistant to change. If the author or right holder wants to make a change to the product, they reprint the literary work in a new edition or issue a new version. Similarly, until around 2012, in software development a linear development procedure was followed, starting with an idea, constructing the necessary requirements, shaping its characteristics, coding and designing its composition, testing it in-house and then implementing it on the customer's computer. For this, the developers and software companies worked with major release versions they distributed to the users on CDs.[1377] Minor updates were provided to the user through free online releases or separate physical mediums. When the software at some point reached a complexity that the system could no longer handle, and the software got to the end of its life cycle, a new major version of the product had to be released.[1378] 758

As explained, copyright is used to work with stable and fixed goods.[1379] It was therefore natural for lawyers to apply copyright to software developments, then still developed linearly, because, similarly to books, for over three decades software was released on a fixed medium and only slightly adapted afterwards. However, thanks to the new server-based software commercialization,[1380] software programs today can be adapted, expanded or partially downsized with a few clicks. Consequently, there is no final and stable version 759

1376 For more information on the international statutory developments, see Chapter 4 Section III.

1377 See N 382.

1378 See N 134 and N 435 ff.

1379 See Burkert/Hettich/Thouvenin, 56 ff.; Nimmer/Nimmer (2016), N 2-33 f., with reference to 17 U.S. Code § 101.

1380 See N 182, N 202 bullet point 3 and N 419 f.

of a computer program with which copyright would be consistently used. The question discussed here, therefore, is how these newer development trends can be integrated into classic copyright law, focusing solely on this legal institute, as patent law does not have the same problem.

760 There are two examples that may help to illustrate the discussed issue. It has already been mentioned that alongside linear development[1381] there are also *spiral development methods*.[1382] Around the turn of the millennium, Ken Schwaber and his colleagues revived and significantly enhanced a revolutionary method called *Scrum*.[1383] The method is not only iterative, but also incremental and flexible. The software product can be developed piece by piece or successively but is often also delivered to the customer in modules, starting with a viable product as soon as it is ready instead of waiting until the whole software product is finished, assembled and tested. The Scrum method, today, is one of the most discussed and applied approaches in software development. Most of the interviewed companies followed this method.[1384] Also, according to a 2019 report from SwissQ, currently over 68% of all Swiss companies use an agile approach to develop their programs.[1385] Meanwhile, the results of the Standish Group's Chaos Report suggest that agile projects have up to four times the success rate of waterfall projects for all sizes of projects.[1386] Today, as time-to-market[1387] becomes increasingly important for market players, providing a functioning first version to the customers sooner and including the users' feedback in the further development process is crucial, and can determine the success of a product.

761 As software solutions have become the backbone for most industrial and service processes, the operating environment has to be functioning and running 24/7. While many technical possibilities have been discovered, the complexity and expectations of technical solutions and their capacity have increased accordingly. If a program malfunctions or a modification needs to be made, the software has to be adaptable at any point of the day from wherever the user is.

1381 See above N 172 ff.

1382 See N 175 ff.

1383 Schwaber introduced the Scrum approach at a conference in Oopsala (see SCHWABER) and then discussed the method with Mike Beedle in their book, Agile Software Development with Scrum, in 2001. See also N 175 ff.

1384 See above N 380 ff.

1385 SWISSQ, 18 and 32.

1386 STANDISH GROUP, 7.

1387 For more information on the 'time to market', see N 198, N 388 and N 498.

Software companies implement new functions and features, updates and bug fixing on a continuous basis from miles away. This approach is called *continuous delivery*.[1388]

In both approaches – incremental development and continuous delivery – the final expression is altered after its first release. With incremental development, new components are *added fragmentally (incremental extension)* to the first released version, while in continuous delivery, *inner enhancements* are conducted through modifications. The fact that the individual software versions are repeatedly changed in substance and thus remain dynamic, constitutes the essential difference between these approaches and a linear method from a copyright point of view. Even though both newer approaches are widely established in software engineering, their integration into intellectual property law has, so far, largely fallen short. Lawyers in practice try to overcome this problem of a missing solution in copyright by addressing the issue in contracts.[1389] But due to its abstract and dynamic features, software is a good that is difficult to capture in the static boundaries of a contract. Contract law therefore does not fully meet the needs of the practitioners. It may help to assign certain risks within a contractual relationship but it cannot settle the intellectual ownership of the software product. Further, this does not solve the problem for non-contractually regulated situations. It would therefore be desirable to find a solution to the problem within intellectual property law. 762

On an international basis, copyright remains the most important institution in the field of software development and commercialization. At the same time, the interview series has shown that copyright law needs to be partly revised or re-interpreted in a modern way in order to remain applicable to newer software trends.[1390] It is, however, unrealistic to expect that the international copyright framework can be regulatorily adapted in the near future. The following two sections will therefore try to outline a solution to the problems within the existing setting of copyright. 763

1. Incremental Extension

Incremental extension or fragmental addition refers to the circumstance that in incremental development the components or new features such as add-ons 764

1388 For more information, see above N 182 ff. and N 380 ff.

1389 See good legal integration of agile and iterative development methods by means of contract law, in Straub (2015).

1390 See N 492 and N 651.

or new functionalities are provided to the users one by one instead of in one complete major version. This is subsequently symbolised with Figure 10, in which "Additions" are added to the "original".

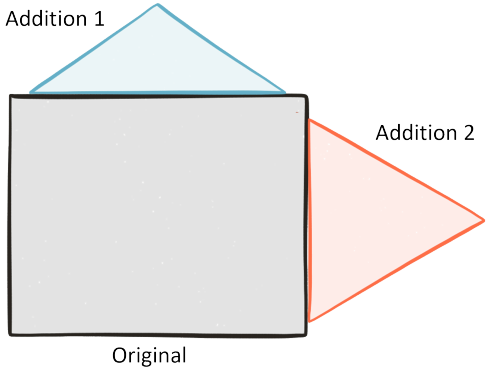


Figure 10. Incremental Extension (Source: own illustration).

765 This conflicts with the traditional approach in copyright where the complete and final work is published on a medium and distributed on the market. There seem to be four possible ways to resolve this issue.

766 First, one could say that even in this scenario only the final version of a software product is protectable. This would mean that, regardless of the time sequence or the form in which the developer delivered their product, it would only be when the product was assembled and declared as complete that it would be protected under copyright. The difficulty with this approach is that, as most software companies add new fragments to their software systems or provide new functionalities or fragments every once in a while, the right holder would be left unprotected until the product was completed and combined in its final stable version. This could weaken the original right holder's position in case of contract negotiations or infringements and the ownership structure of the software would be complicated unnecessarily. Therefore, this would not seem to represent a reasonable solution to the problem.

Second, we could adapt the original understanding of copyright according to which the first created work follows some kind of stable structure and can, thus, not be modified. The copyrightable version would therefore be caught in the first software creation. Every later delivery of a component in this scenario would not affect the copyright scope of the protected work and would therefore be insignificant for copyrighting. They would be free to use. Under certain circumstances, they would be covered under the author's right to edit the original, but newer components could never become copyrightable themselves. Although this approach would comply with the classic understanding of copyright, this solution would bring more disadvantages than benefits. The original function of copyright to work as an incentive for creators would be limited to the creation of the first version. This would disadvantage dynamic models such as incremental development and continuous delivery. I thus would not recommend this approach. 767

A third approach would be to grant copyright protection for each component created. We would start with the first version created and later released to the market. If this part fulfilled the requirements for copyright protection, it would be copyrightable (work1). The second part of a component would then be tested on its copyrightability again. If it fulfilled the requirements, it might be considered as a stand-alone work with individual components (work2). This process would then be repeated for every new part of the software that was created. In the end we would have a set of works ($work_1$, $work_2$, (...), $work_n$, which are combined into one big software composition that the end user utilizes. 768

$$work_1 + work_2 + work_3 + work_4 + work_5 + work_n$$

From a legal perspective, however, the single deliveries would be distinguishable separate works, treating every creation as its own copyrightable work.

In the fourth and final approach, the first version is already considered as a copyrightable work as long as it fulfils the requirements (work1). When a further extension is available, the provided parts are examined for their copyrightability. If the extension does not fulfil the copyright requirements, it cannot affect the copyright features of the first software version created, positively or negatively. However, if it is able to meet the requirements, the originality scope of the original work is extended with the new characteristics of the new elements (original strength of release2). The originality of the work contained in the new extension is therefore added to the first one's protection scope. Together they form one new copyrighted work ($work_1$ $release_2$). If a fur- 769

ther element is available, it is tested on its copyrightability again. If protection is granted, its contribution is simply assigned to the originality of the software product that was available prior to the new version ($work_n$ $release_{n\,1}$).

$$work_1 + \text{original strength of } release_2 + \text{original strength of } release_3 \ldots + \text{original strength of } release_{n\,1} = work_n + \text{original strength of } release_{n\,1}$$

Simplified, this means that the copyright scope of a software product can be expanded with every new original component that is provided. The individual parts are not treated separately but are considered as one work that is simply gaining new characteristics. The original strength of later releases is assigned to the original work release. In this way, software can be extended and the product remains fully protected.

770 The third and fourth approaches both enable copyright protection to be available for every new element as long as it fulfils the requirements for copyright protection. They overcome the disadvantages of the first two models, whereby it is either only the first or only the final version of a software product that is protected. The third approach focuses on the individual parts and extensions and disregards the fact that every element belongs to one software project. The fourth approach, on the other hand, ignores the fact that the different fragments were not created at the same time but separately. One major contribution of the final approach is that it reflects the technical reality that ultimately all the components are part of one software work which all collaborate within the same software system environment. The third approach has the advantage that the contributions of each new feature may be captured separately in economic terms and that their importance for the whole software solution can widely vary. While it may give the right holder more flexibility in managing his or her copyright over each piece of work, it also becomes more complicated. The third model may therefore become too theoretical and hard to handle in practice, as the software components only work within their implementation environment and usually cannot be separated and commercialized out of their usual context.

Incremental extension or fragmental addition can therefore be implemented in copyright in four different ways. I would recommend applying the last approach and accepting the first version as a copyrightable work – if the requirements are fulfilled – whose protection scope is expanded with every new extension. Whenever a new component is delivered, the work receives new characteristics and elements. The copyrighted object thus becomes a dynamic composition that is open for new features. 771

2. Inner Change

The term inner change refers to the circumstance that, nowadays, a published single piece of software, such as a component, is kept under continuous construction.[1391] Given the fact that copyright usually works with creations that, once published, remain identical, works that are altered in substance within their original boundaries are non-systematic for copyright; it is tricky to determine the protection scope and content of a work that keeps changing its shape. Unlike incremental delivery, in continuous inner change the outer protection scope generally remains the same as the changes appear within the existing mantle, where a part of the substance is modified or replaced. In practice we can also observe changes in or on a software product that combine the scenarios of adding a new piece to the original and changing some of its inner content at the same time. For practical reasons, I have excluded that aspect from this section. 772

Where software can be changed continuously the question arises to what extent the changes affect the substance of the work's copyright protection. In order to evaluate this problem, we have to determine the work's previous composition. One can look at the specific section to be altered within the original and determine how important it was for the expression of the original, the creative contribution it offered and to what degree it showed originality of its own. Then, we would observe the new composition after the alteration and evaluate how it affected the work as a whole and particularly the part we wanted to change. Thus, the individual versions of a software product are important, as each must be examined for their originality. There are three possible scenarios (Figure 11): 773

1. the alteration represents an enhancement of the previous composition;

[1391] See, for example, the case of continuous delivery in N 182 ff. and N 380 ff.

2. the alteration offers little or no enhancement of the previous composition;
3. the alteration has reduced the particular part's contribution or original strength and may, ultimately, reduce the copyrightability of the entire work.

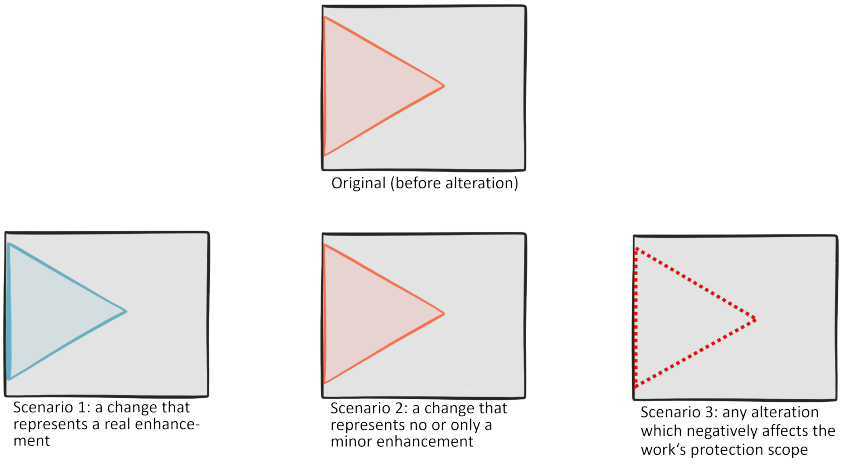


Figure 11. Different Scenarios for Inner Change (Source: own illustration).

774 According to the first scenario, where a change represents a real enhancement to the previous composition of the altered section or even to the whole work, the outer mantle of the protection scope remains the same as before because no additional component was added. However, the originality of the respective section and thus of the whole work intensifies. Consequently, the work character is improved. The improvement of originality is so remarkable that, from a legal perspective, the substance of the previous copyrighted work may be regarded as replaced. The work's individuality or creativity has been strengthened.

775 The second and probably more frequent scenario is if an enhancement to one part of the work only has a minor positive effect on the work's copyrightability, even though it might represent a technical enhancement or improve its usability. In this case, the improvement is not able to meet the required creative step to increase the work's originality. Hence, the work's substance has changed but the extent of protection has not. This may happen if either the altered section is not significant for the work overall or if the particular alter-

ation has only little or no effect on the section's originality. Again, the copyrighted scope is not expanded because no additional components have been added to the work itself. The whole change appears within the original mantle of the work.

The final possibility occurs if a change in some way causes deterioration of the work's substance. Although it may represent an enhancement from the project's point of view, the work undergoes a negative change in terms of copyrightability. This may happen if the previously provided degree of originality cannot be maintained when altering the respective section. At worst, the originality not only of the section but of the whole work is affected in a way such that the work loses its overall creative expression. In this case, the copyright protection of the entire work would be destroyed. This could, for example, be the case if an engineer replaced their own original solution to a problem with a better available ready-to-use component, without using any creativity in implementing it. 776

Consequently, if a work's outer boundary remains the same but changes are conducted inside the given structure, it should be kept in mind that the copyrightability of the work might be affected. Depending on whether the work's originality increases, decreases or is left unaffected, the copyrightability of the work may be supported, remain similar or deteriorate. Working with server-based commercialization models and continuous delivery may thus have a positive or negative effect on the software's copyrightability throughout its development and commercialization. 777

F. Conclusion

This chapter picked up certain aspects of the legal and technical foundation and discussed the findings of the interview study regarding software's protection scope in copyright and patent law, such as the subject matter and the relevant protection requirements. I then suggested several small improvements to the definitions of the established protection criteria that would better capture software as a creation in IP law, but could also be adapted for copyright and patent law in general. 778

A set of different software components was discussed to evaluate these elements' eligibility for copyright and patent law protection. To date, computer programs have mainly been understood as an intellectual good or process that 779

is expressed in a linguistic command form.[1392] The visual appearance of software, for example in the user interface or the look-and-feel, has tended to be neglected. Ideas for solving certain problems are largely limited to the field of algorithms in patent law, if permitted at all. This suggests that it is still difficult to truly define computer programs. It is suggested that, in future, functional elements of the look-and-feel as well as functions and features should also be integrated into patent law alongside algorithms. Further, the previously narrow scope of copyright law should be applied to visual elements in a user interface and the look-and-feel and further expanded to the development documentation. Structural and organizational elements in the source code should be accorded greater consideration in evaluating the copyrightability of program components.

780 With the help of the interview findings, it was also discussed to what degree technical necessities, engineering standards and business best practices are covered under IP protection, and how copyrightable expressions can be distinguished from non-copyrightable ideas in software development. Based on the established merger doctrine, abstraction test and the current scènes-à-faire practice, a rule-of-thumb method called the blackbox test was proposed to assess software's eligibility for IP protection.

781 In the final section, the current trends in software development – incremental development methods and continuous delivery – were analysed from a copyright perspective. It was discussed how, due to their dynamic characteristics, these two development methods contradict the classic comprehension in copyright of a work with static and fixed features. It was suggested that incremental extensions or fragmental additions to and changes within previously released works could be integrated doctrinally into copyright, without having to request any regulatory revisions.

IV. Term of Protection

782 In copyright and patent law the term of protection refers to the period between when a subject starts being covered under the protection umbrella and protection ceases.

783 Today, at least with registry IP rights, the term of protection in IP law usually starts when the property and its merits are first disclosed. In the case of de-

[1392] Similar conclusion in: Boecker, 56; Hilty/Geiger (2011), 154 f.; Hilty/Geiger (2015), 616 f. and 619 f.; Jersch, 192.

sign rights, patent law and trademarks this commonly happens when the registry receives the application and gives the applied subject a personal application date. For copyright, the term of protection starts with the work's creation. No formalities are requested. During the running period, the IP is protected from external interventions as the right holder obtains lead time to get established on the market. After a certain period of time, the protection expires. This point in time may either be fixed, for example with regard to a specific maximum term of protection after the good's disclosure or release, e.g. in patent law, or be indefinite in advance because it is tied to a certain later event, such as the author's death in most copyright systems. As soon as the term of protection has expired, competitors and society are free to use the intellectual property.[1393]

This chapter is split into two parts. The first part discusses where the starting point for copyright protection is set in the current legislation, exploring the difficulties this entails for computer programs and how these problems could be addressed without further legislatory changes. The second part focuses on the end of the protection term, called expiration, and compares it with the life cycle of an average software development. It is outlined where problems in the current statutory approach lie and what the legislators' possibilities are to adjust it. 784

A. Starting Point for Legal Protection

As mentioned, the starting point for legal protection is when a software component is first granted shelter under the mantle of IP law. In patent law, the protection starts when the registry receives the application. There is thus a fixed, identifiable point in time when the protection arises. In copyright law on the other hand, protection starts automatically, without any formal act such as an application process, with the work's creation. However, the circumstances when a sufficiently qualified work can automatically obtain protection as a "creation" must be determined separately for each object. The difficulty of determining a starting point for protection therefore only arises in copyright law, which is why the following section will focus solely on this legal institute. First, I will discuss the particular difficulty with the classic start for protection in copyright concerning software. I will then introduce a different approach for tackling these difficulties in a separate section. 785

[1393] The legal foundation to this was introduced above; for patent law see N 330 ff. and N 336 ff., and for copyright N 367 ff.

1. The Problem

786 Looking at the time sequence of intellectual property protection, it appears that in setting a starting point for legal protection the legislator was orientated along the linear development phases model. The lawgivers seemed to have an idea of a process in which, during development, somebody had a creative idea, an impulse for a new project, that was then specified into a concept, afterwards became the final product through coding and designing, before being implemented definitively into its future system. Any additional technical interventions from this moment on were only necessary for minimal maintenance purposes.[1394]

787 Further, if one compares this chronological sequence with the protection subjects in copyright, it can be seen that the institute is orientated towards the end product of a development project: In software, it mainly shelters the visual expressions in a user interface or look-and-feel and the source code, all representing extensively developed components of the final product.[1395] This gives the advantage that the legislator and society have something tangible to observe and look out for, some kind of visible barrier between when a good is not sheltered by IP law and when it is finished in its definitive form and protection is granted. We can wait for the work's realized creation to grant protection. As this represents a clear point on the time axis, many lawyers support sheltering the end product of a development.[1396,1397]

788 But the subject's increased visibility is also a negative aspect of the current approach. Every time a part of the software is made available to a third party, the accessible know-how used to develop it is disclosed.[1398] And as the density of know-how included in a component grows linearly with the development process, the final version of a source code, which is legally protected, only represents the last step in a long chain of relevant and valuable decisions in the development process. As the software companies outlined, most of their know-how and resources are invested into the development of a detailed con-

1394 See N 172 ff.

1395 Ernst, 209; see also discussion in Lemley (1995), 6 f. regarding concepts and the idea–expression dichotomy.

1396 See for example Rauber (1988), 281; Horns, 5.

1397 The extent to which pure drafts in the form of development documentation are protected by copyright today remains controversial. For more information, see N 356.

1398 See N 526 ff. and N 535 ff.

cept, which is why it often shows the highest creative or inventive strength.[1399] At the same time the interview series also showed that nowadays companies often present and share their prototypes with potential customers, principals, purchasers and investors.[1400] When intellectual property law only protects the final result in the form of an algorithm or source code, all the know-how that has been invested and shown at an earlier stage of development is accessible without being granted any legal protection. Between the point when sensitive know-how is disclosed to these outside parties and the moment at which a source code can be definitively realized represents a time lag during which a creation or invention is not protected but exposed. This means that all the technical finesse, the little tricks and conceptual drafts are unsheltered, creating a *protection gap*. For the product software the classic starting point involves unresolved risks for the developing companies and right holders.

2. Potential Starting Point for Linearly Developed Programs

Intellectual property law wants to shelter the creator's creative contribution. It would therefore be consistent to set the starting point of legal protection where the creative step occurs. 789

In the interview series, the software companies explained that in the development process most of the creativity is invested during ideation.[1401] At the same time, they agreed that protecting abstract ideas might lead to monopolized hatching, which they considered would be bad for creativity. Instead, the software companies suggested potentially starting legal protection when the abstract idea was minimally fixed in a creative concept.[1402] This means we would have to find the point along the time axis where the creative step occurs and is sufficiently materialized to assess its original strength. There appears to be a time spectrum between where most creativity can be observed in the development process (ideation and conceptualization) and when the final product is released. Somewhere along this spectrum lies a point where the idea becomes tangible and sufficiently elaborated for the project to be presented to third parties, such as investors and customers. The interviewees explained that this moment arrives at a different point for every project.[1403] Still, the boundary between abstract ideas and an applicable invention or creative ex- 790

1399 See N 549 ff.

1400 See above N 500 ff.

1401 See N 547 ff.

1402 See N 547 ff.

1403 See N 551.

pression is very vague and even for experts it is difficult to determine whether an idea is developed far enough to be considered a protectable creation.[1404] This thesis can therefore not identify "the all-time valid starting point" but, instead, will try to define the theoretical starting point and narrow down some potential cases of applications. For the individual case, the delimitation still has to be conducted separately.

791 It may help to recall that computer programs represent command sequences that aim to solve a particular problem.[1405] As Singer explains, these commands can be used to express our thoughts (to the computer).[1406] We use computer programs to communicate with the machine. And, similar to human language in a book, the question is how developed the expression to communicate with the recipient – the final wording or the drawings – has to be before we cover it under a protective umbrella. Nimmer et al. analysed the same problem in their paper and had the idea of adapting the *abstraction test*[1407], which is used in U.S. courts to evaluate whether an idea and its expression are merged, to determine whether a development has gone past its abstract stage. In their paper, they suggest that the content of the creation can be observed alongside its temporal development axis. They describe how, at the beginning of a project, the programmers only have a very general notion of what they want to achieve with their program or what it will do. These general ideas represent abstract thoughts that fall under the realm of unprotectable ideas. But by the end, the programmer will have produced code that "is likely to constitute a protectable expression". They concluded that "at some point between these extremes, the level of specification is sufficient to cross the line between an idea and expression".[1408] As in the abstraction test, a layer-by-layer approach is adopted for analysis. Although I like the idea of adopting the abstraction test to determine the starting point for IP protection, unfortunately Nimmer et al. provide no information on how it could be implemented in practice. They do not offer any instructions as to how an examiner should proceed, nor do they suggest which incidents to look for to determine the creation's abstractness or specifiction.

792 Nevertheless, observing the constructive process along the development axis, as Nimmer et al. suggest, in cases where software is developed linearly repre-

1404 See N 551; same opinion represented in: Fisher, 16 f.; Neff/Arn, 140.

1405 See definition earlier in N 124.

1406 Singer, 112 f.

1407 For more information on the abstraction test, see N 727 ff.

1408 Nimmer/Becchini/Frischling, 638.

sents a reasonable way to proceed. As Wittmer suggests, we could evaluate the development's technical advances in the development documentation, which illustrates and provides information on the development process and its achievements.[1409] Again, the abstract idea is not protected under copyright. But after that a component should be open to copyright protection if it is specific enough to exceed the level where ideas are too general and still convertible, and can thus be tested for its statutory eligibility. The form or arrangement of the creation might suggest the potential to be protected, but it should not determine the starting point of the expression. It is thus necessary to search for a moment in time, rather than shape, when the creation's basic structures and elaboration have become more than perceptible, predictable to a certain degree, yet not fully determinable or embodied in a physical form. *The conceptual characteristics of the creation or invention have to be visible enough so they can be analysed as to whether the traditional requirements for copyright are fulfilled.*

This point comes at a different stage for each creation or invention. In a creative work, the expression is evident when considerable and substantial details or additional particularities are exhibited, through which the author offers their personal contribution to the realization of the idea in a practical way.[1410] If an expert cannot evaluate whether the requirements for legal protection are provided in the creation, the subject is still *too abstract*. On the other hand, if a creation is further elaborated but it does *not additionally strengthen the originality* of that creation, this means the creation has already gone past the creative step to become a shelterable intellectual good. Likewise, in the course of development, the associated development documentation for a software project may already include some elaborated concepts for software components which are tangible and hence eligible for copyright protection, if it offers enough substantial details and original strength.[1411] 793

The interview series further showed that there are certain development concepts that can be implemented in many different ways, and that for these the exact mode of realization does not matter because the overall creative concept is decisive.[1412] For example, we can cite Singer's example of an original pseudocode, for which it is not important how the code is technically imple- 794

1409 See WITTMER, 34.

1410 See also above N 454 ff.

1411 See also discussion above in N 719 ff.

1412 See N 463 ff.

mented in a specific programming language in the source code.[1413] The implementation rather is routine work. The contribution lies in the creative concept for the pseudocode, including the detailed description of its behaviour and practical functionality in a concept or draft. These goods thus reach the creative step at an earlier point in their development. On the other hand, if the main idea is to improve a technical process or offer a better performance, the actual coding of a slim and efficient implementation commonly represents the IP's main contribution.[1414] In this case, the creative step – and thus the starting point for legal protection – shifts towards a later development stage where most of the labour and effort occur and the original creative contribution of the author becomes visible and more tangible, for example in the case of an enriched source code. The starting point may thus vary depending on the subject. It is consequently important to identify the particular creative step for each good individually.[1415]

3. Potential Starting Point for Non-Linear Development Approaches

795 Finding an appropriate starting point for copyright protection within the linear software development process is very difficult. It becomes even more complicated if a creation has been developed with a non-linear method, such as:

- the *spiral method* where a project is realized in a circular form and conceptualization and realization are repeated for every new project module tackled; or
- *inner change* in which the outer mantle of the creation remains the same but changes are made within the mantle.

796 The question in these complex projects is where the earliest starting point of protection lies and how new work components are integrated into the running protection term.

a) Spiral Development

797 In spiral or iterative development, the project does not follow a straight linear development where in the end we obtain a complete software product but in-

1413 SINGER, 112 f.

1414 See N 480 ff.

1415 See similar argument in NEFF/ARN, 140 ff.

stead the project evolves in a circular way and each phase is repeated for each module or functionality that is developed before the product's final release. The problem in determining the starting point for spirally developed software lies in the circumstance that it usually takes several years until the work, as a combination of several modules, obtains its final shape and is published as a complete product, while conceptualization and realization occur over and again in the course of the project. If we assumed that we could only evaluate the copyrightability of software in its full and definitive arrangement, the software would be left unprotected when it was first presented to third parties in a first (but mostly completed) set of modules.[1416]

In addition, the actual start and finish of an iterative project are difficult to de- 798
termine. Usually a longer time frame is required to develop a set of first modules because the software company commonly starts with a few more important functions and then develops other modules that complement the previously developed ones, such as in incremental methods. Depending on the complexity of the module worked on, its system environment and the exact method followed, the interviewed companies said that one module (including at least one whole spiral cycle) could in theory be realized – developed, integrated, reviewed and adjusted – within a sprint of one to four weeks.[1417] Although an individual module would function autonomously, there is a certain threshold that has to be overcome, a *set of first modules* has to be formed, so that the basic characteristics of the desired software product are recognizable and the good becomes able to function properly and achieve its predetermined goal. Before that moment, even after conceptualization of a first few models the further course of the software project is (still) unresolved and relatively open. The same applies to the other software elements to be added in further repetitions. But when the threshold of sufficient concretization is reached, the development becomes minimally fixed and autonomous, a minimum viable product.[1418] In practice, this first relevant set of modules regularly represents the prototype that a project manager could then present to third parties, such as investors and customers. According to the rough indications of the interviewees, a first prototype of their products and services could be developed within an average of two years.[1419]

1416 For more information on commissioned work and project pitching, see above N 500 ff.

1417 For more theoretical background on the spiral method, see N 175 ff.

1418 For more information on how the spiral development method is applied in practice, see above N 380 ff.

1419 See N 391.

799 That third parties are able to make business decisions on the basis of the prototype implies that the prototype usually includes the main functionalities that enable the final software product to become perceptible and forseeable for their observers. Already at this early stage of the development, the main characteristics of the creative expression are preliminarily fixed. *The prototype functions as a materialized, solidified outline for the further project, which can also be tested as to whether it is able to fulfil the copyright requirements.* It therefore seems reasonable to determine as the starting point for legal protection of a spirally developed good the stage when the first prototype is ready. To start the legal sheltering at an earlier stage, for example with the concept of a first developed module, which may be changed several times in an iterative approach, does not appear expedient. It gives no direction for the further modules that still have to be built. An important exception to this is if an overall concept for a software project is sufficiently detailed to reveal the shaping characteristics not only for the first project modules (resulting in the prototype), but also for all other later modules too. If this concept exhibits originality, and thus fulfils the copyright criterion, it should already be protected from this point in time, for example in the form of development documentation. However, this is unlikely to ever apply to a spiral development (outside the linear waterfall model). In general, waiting for a later stage in spiral development than the prototype also does not seem useful as usually the first few modules are the ones that are economically significant for the product, which is why they should be put under a protective umbrella as soon as possible in order to close the protection gap. The later added modules usually only complement the overall product and may influence a work's degree of originality, but most often do not have any effect on a work's term of protection. Consequently, the tradeable prototype should in standard cases be regarded as a starting point for legal protection in spiral development.

b) Inner Changes

800 Inner changes refer to alterations within a product's existing mantle after it has been released. Under this term we also include updates and structural or visual changes, particularly where sections or elements within an existing work are altered.[1420] While the contours of the work remain the same, its substance is manipulated. The question is how such changes within the work affect the original's protection term and to what degree they trigger a starting point for a new term of protection.

[1420] See for example in the case of continuous delivery, N 182 and N 382.

Previously, we discussed how and to what extent changes within a software product can affect its copyrightability, the protected scope of the software. We distinguished three different types of changes: first, an alteration that represents an enhancement of the previous composition; second, an alteration that offers no or only little enhancement to the previous composition, and third, if an alteration reduces the particular part's contribution and may, ultimately, negatively affect the copyrightability of the entire work.[1421] The same scenarios will be used for the present discussion. As these offer three very different applications, the term of protection again has to be determined for a particular work and change in question. 801

In most cases, changes within the work will not affect its term of protection, as the outer mantle of the work remains the same. Copyright protection will instead start with the original release and keep on running. This is what was referred to with the second scenario, where a change within the work offers little or no enhancement to the previous work quality. Even though the substance may be altered, the overall creative appearance usually does not change to the extent that the work is perceptibly different as a whole. Therefore, although the appearance or structure of the work is manipulated, the protection term remains unaffected. 802

Only in very rare cases with significant improvements in substance, is the altered part able to form a different work by adapting inner parts without adding anything from the outside. This is referred to in the first scenario. It is almost as if the new work represented a shelterable secondary work of the original, an independent version of the product. The original and the new work are similar to some extent but the new work is able to distinguish itself by exhibiting originality of its own. If a change in a work reaches this standard, linking the altered section's protection term entirely to the original work would be illogical. As the new composition distinguishes itself substantially from the original, its term of protection should at least to some extent be valued autonomously by giving it its own starting point for the protection term. However, this special situation where a section can obtain a full new term of protection through an enhancement is exceptional and should therefore only be granted in particular cases where the manipulation effectively represents an entirely new work within the existing mantle. 803

We also introduced a third rare scenario in which a planned alteration within the inner structure of a work was not able to replace or maintain the previous 804

[1421] See above N 772 ff.

substance but, rather, had a negative effect on the work's copyrightability. According to this understanding, such a negative change in substance would not be able to influence the starting point of legal protection of the work because the protection started automatically with the first creation and the latter changes do not affect the work's outer mantle. Deteriorations may however have an effect on the copyrightability of the work itself, and as such on its protection, because if a product is no longer able to fulfil the legal requirements for copyright protection, by losing its originality, it could forfeit its IP protection before actually reaching its expiration date.

805 Consequently, if a work's outer boundary remains the same, but changes are conducted within the inside of the given structure, we can generally assume that the term of protection is not affected by the change. Only if the change qualifies as a real enhancement that exhibits its own originality, or, on the contrary, if the alteration represents a relevant degradation that affects the copyrightability of the work overall, will a change within a work become relevant.

B. Expiration

806 Expiration refers to the point at which the particular exclusionary right is lost due to lapse of the term of protection. In patent law expiration is set at 20 years after the patent's official application date. In copyright, computer programs are protected for a minimum of 50 years after the death of the last author involved in the work's creation. These terms cannot be extended and are thus referred to as maximum periods.[1422]

807 The legal problem of the expiration period is two-fold; on the one hand, it is difficult to find and set *an appropriate term of protection*. On the other, a *specific event* has to be selected to which that term of protection can be tied – a connecting factor that triggers the maximum period. These two problem areas are discussed below.

808 The outlined questions would fill a doctoral thesis of its own, which is why the present thesis can only offer scientific data to show the potential for discussion and encourage further reflection. This section will therefore discuss what was found in the interview series and circumscribe potential approaches based on these findings.

[1422] The theoretical background to this section has already been introduced under the respective type of intellectual property: for patent law N 336 ff., for copyright N 369.

1. The Duration of the Protection Term

Today, the protection period is set so long that, in fact, there will hardly ever be an interest in using the protected software beyond these time periods.[1423] In view of the dynamic and short-lived software industry, the question arises as to whether the difficulties associated with absolute property rights in software protection could be alleviated by adjusting the term of protection. Troller likewise discusses whether, if the general public is to share in the benefits of a creation in exchange for granting a monopoly-like right, its term of protection should be set such that it ends before the inventive creation is completely overtaken by technical developments.[1424] 809

A patented part in a computer program is protected for 20 years after the patent was applied for. Where copyright protection is applicable, a work of software is protected for up to 50 years after the death of the last author, irrespective of the date of its creation during an author's lifetime or release. Considering that the average life expectancy in the United States and Europe is around 80 years,[1425] and assuming, for example, that the developer had introduced their product at the age of 25, copyrightable computer programs could be protected for a period of over 105 years. As mentioned, probably only one computer program has ever reached the end of the protection term, namely Alan Turing's machine Christopher.[1426] This was possible because Alan Turing ended his own life in 1954 at the age of 41.[1427] The computer program he created during World War II was left to the public domain in 2004. Only partially 810

1423 See N 558; see a similar statement in: Edvardsson, 32; Straub (2002), N 10; Straub (2011), N 148; Dessemontet, N 967; Marly, N 363; Neff/Arn, 314; see also discussion in Troller (1983), 119 ff.

1424 Troller (1983), 119.

1425 For the United States the average life expectancy is 78.6 years, within the EU the average life expectancy is 80.9 years, and in Switzerland it is around 83.6 years (see OECD 2017 statistics on life expectancy).

1426 For the previous discussion, see above N 499.

1427 Under the British Foreign Office, Alan Turing developed an algorithm that was able to decode secret messages of the Nazis. By implementing this algorithm into a machine, the Allies were able to decode daily messages faster and more reliably. As the first in history, Alan Turing's machine was capable of being altered to not only solve one specific task but rather a set of problems, as long as the problem could be expressed in the form of an algorithm. The program was thus applicable in a very dynamic way. His inventions motivated scientists to advance his studies to the next level, forming the basis for today's computers as we know them (for more information about Alan Turing see: Copeland; Randell; Newman).

comparable to this development is the commentary by Ada Lovelace from 1843, which contained an extensive concept for the programming of a machine by Charles Babbage. From a historical perspective, Lovelace was partly (and controversially) described as the first programmer in the world. From today's point of view, however, the detailed document should probably be regarded as a rough concept for a computer program rather than a copyrightable version of a computer program itself. The idea for Babbage and Lovelace's first computer could only be put into practice in 1960 with the computing capacity anticipated by Babbage, almost 110 years after Lovelace's death.[1428]

811 Independently of how it is captured by law, every good shows some kind of life cycle.[1429] As Lemley and Burk emphasize, the life cycles in software engineering are shorter compared with those of other technologies.[1430] According to the interviewed representatives of software companies, currently a computer program's life cycle is estimated to be between 5 and 25 years, with an average term of 10 years before a system has to be disposed of entirely.[1431]

812 Comparing the granted 20-year protection after filing, in patent law, and the 50 years of protection after the last author's death, in copyright, with the estimated program life cycles of 10 years on average, it raises some questions. The legal term of protection may potentially be up to ten times the product life cycle of a software product, for example in terms of copyright.

813 From an IP lawyer's perspective, this difference seems significantly imbalanced. But how did we get here? The original idea of a long protection term in copyright was to perpetuate the creator's relationship to their creation even after their death, to preserve a legacy.[1432] However, computer programs do not represent classic figures of art but rather technical goods which nevertheless involve creativity and inventiveness. It is therefore questionable whether a technical creation deserves to benefit from the long protection term granted in copyright law for artistic works.

814 On the other hand, copyright and patent law are designed to ensure that the creator has the possibility to earn back his or her investments and make some

1428 For more information, see FUEGI/FRANCIS, 25.

1429 For the technical background to life cycle development models see N 134, and for the practical value of software life cycles, see N 435 ff.

1430 LEMLEY/BURK, 90.

1431 For more information see above N 435 ff. and N 437 in particular.

1432 DESSEMONTET, N 968 and 70; see also discussion in Commentary to the German UrhG (Katzenberger/Metzger), § 64 N 1 ff.

profit during the 20 or 50 years or more of the protection term. The author has an interest to decide on and market the creation. Samuelson et al. note that in a perfect world where the legal system is market-orientated, a computer program would be protected against commercial interference "just long enough to enable its developer to enjoy the same lead time as other innovators who contributed equal value to the market".[1433] It emerged in the interview series that a computer program is marketable within approximately 3 years and that it has to be replaced with a new system after a maximum of 25 years, and 10 years on average.[1434] The right holders consequently have approximately 7 years to earn back their investments. Furthermore, in the software manufacturing and distribution market most products are not published on a physical medium but instead are duplicated digitally and distributed online. Through these simple improvements, the economies of scale in the software market have increased substantially.[1435] The time frames required for a developer to earn back his or her investment have therefore shortened with the rise of digital distribution models during the last decade. The software companies stated that a software manufacturer should be able to get established on the market and earn back their investments within the first few years of lead time, otherwise the chances of succeeding later in the market are quite low.[1436] This statement is all the more relevant at a time when the time-to-market is becoming more important and working in a right holder's favour to more easily commercialize their products.[1437]

Even if an extensive term of protection of more than 50 years after the death of the last developer does not hurt anyone, it nevertheless represents unequal treatment of the interests involved and causes a form of *overprotection*. It does not take into account the expected lifespan of a computer program; every year of protection that is granted in addition to a program's lifespan represents an artificial monopoly for the right holder in the form of additional lead time.[1438] Or as Harison puts it: The current term of protection encourages the right 815

1433 Samuelson et al., 2413; see also discussion in Dutfield/Suthersanen, 109 ff.

1434 See the findings in N 437.

1435 See N 189 ff.

1436 See N 561.

1437 See also N 498.

1438 See a similar discussion in: decision of the Court of Justice of Geneva of August 6, 1986, published in SMI, 1987, 217 ff.; Von Lewinski Silke, 724; Reichman, 2547 f. and Rauber (1988), 280; see also economic analysis in Scotchmer (2006), 107 ff. and 117; Dutfield/Suthersanen, 109 ff., in particular 110; Commentary to the German UrhG (Katzenberger/Metzger), § 64 N 1 ff.; Landes/Posner (2003), 213 f.

holders to use their invention as a "cash cow" without investing into further development to build on it.[1439] At the point where the development finally falls into the public domain, the technology is usually outdated and not applicable for a modern computer.[1440] The duration of software protection appears to be arbitrary and inefficient.[1441] The interviewed companies further stressed that *the know-how exchanging function of IP would thereby be widely forfeited.* For this reason, the terms of protection in both copyright and patent law were found to be way too long, relative to the life cycle that software products actually exhibited.[1442] It would thus be reasonable to set a (new) particular protection term in patent law and copyright for software products only.

816 In the interview series, the companies were also asked which factors they considered important in determining for how long a computer program should be legally protected. They named three main points of reference: the author or company's will, the time during which the product is still purchased and bought by the users, and the estimated term during which a software product stays technically useable.[1443] Although all the suggestions have their pro and cons, there are certain arguments that clearly speak against or in favour of each.

817 The first suggestion was that *companies could decide for themselves* how long they wanted to shelter their product with IP. The main reasoning was that it is the author or right holder that puts their investment into the development or purchase of the product so it is he or she who should be the one to make the judgement call. In this way, investors would be more willing to spend money on development because they receive a bigger incentive. Although this is a legitimate request, this change in legislation would grant the software companies an extraordinary amount of power and would not sustainably reduce the already very long protection period of software that currently exists. They would be able to make their decisions solely based on their own business. In

1439 Harison, 14.

1440 See N 558; see a similar statement in: Edvardsson, 32; Straub (2002), N 10; Straub (2011), N 148; Dessemontet, N 967; Marly, N 363; Neff/Arn, 314; see also discussion in Troller (1983), 119 ff.

1441 See also mathematical approximation and conclusion in: Scherer, particularly 427, with reference to Nordhaus. Both authors offer interesting ideas on how the term of protection should be set in relation to other factors, such as the investments and social gains, without, however, giving more precise information as to the duration.

1442 See N 437 and N 561.

1443 See above N 554 ff.

the worst case, they would not have to surrender their invention or creation into the public domain at all. This could endanger the balance of interests in intellectual property law disproportionally. It would also significantly contradict the main principles in Art. 5 para. 2 and Art. 7 para. 1 RBC, according to which the term of protection persists independently of the right holder's influence. Instead, the right holders should be satisfied with a clearly determined and forseeable lead time during which they can profit from an exclusionary right and earn back their investments.

818 The second suggestion was that a product should be granted legal protection as long as there is still *user demand*. The supporters of this argue that as long as users are willing to spend money and time on using a software product, the program should be legally protected. They referred to certain computer programs, games in particular, that profit from nostalgia or have been hyped for several decades, similar to best-selling literature or hit records. According to some of the software companies, software manufacturers who manage to create a computer program that is so successful should be allowed to continue to profit from their creation. What these companies did not consider, however, is that the end of a program's protection term does not automatically mean that the product cannot be offered and sold on the market anymore; it is just the right holder's exclusionary right that expires. Similar to the first suggestion, the lasting utilization of a computer program does not take into account that in exchange for the legal monopoly-like right society wants to obtain an insight and, at some point, also full access to the good. With this suggestion, a prevailing product would rarely fall into the public domain at all, even if only very few users continued to buy the product in the end. Again, the balance of interests would be affected. In addition, this factor would be very difficult to measure and control in practice. For this reason, I would not recommend the continued utilization of the software as the decisive factor to determine the term of protection.

819 Most of the interviewees were in favour of an option which I personally also support. They instead recommended linking the maximum term of protection to the *usability* of the computer programs. The software companies argued that after some years a technology would become outdated and more difficult to apply. At the same time, for most users the software is less easy to use. From the company's perspective, at this point in the product's life cycle more money has to be spent on keeping the product up and running. For the same reasons, competitors start to lose interest in copying or imitating the software. The software's usability – its average life cycle – would represent a comprehensible and reasonable point of reference to set the maximum term of protection for

software products. If being set at, for example, 10 years after it was made available to third parties this would give the right holders a good lead time to get established on the market and regain their investments, and their economic interests would be covered. The lead time would also offer them a considerable incentive to keep on maintaining and improving their product so that users keep on spending money on the product or service. The good would fall into the public domain while it was still technically applicable and of use. For all these reasons, I would recommend taking the average usability of a computer program to determine the term of protection. Thus, all the important interest groups – the right holders, the users, other market participants as well as investors – would have an adequate and reasonable incentive to participate in the IP model.

820 According to the findings of the interview series, most of the companies favoured a lead time of around 10 years. If the users' needs and the technical requirements in the field of application were particularly stable, supporting longer usability of the software product, the interviewees believed that a longer term of protection of around 20 years could also be justified.[1444] On the other hand, they argued that IP rights which included more severe exclusivity as well as more market advantages should only be granted a lead time of a few years, for example 3 to 10 years.[1445] According to this argument, copyright protection, being the less exclusionary IP right and protecting wants to protect finished and constant forms of expression, could be provided for around 20 years after its creation or having been shared with third parties. Meanwhile, patent law, which by its nature is a stronger IP right that grants the right holder a monopoly-like exclusionary right for their dynamic and easily editable invention, could be permitted for a shorter period of up to 10 years after the application has been filed.

821 The suggested duration of 20 years for program protection in copyright and 10 years in patent law contradict the current international regulations in Art. IV para. 2 UCC, Art. 7 para. 1 RBC and Art. 33 TRIPS as well as the EC Copyright Term of Protection Directive.[1446] Due to the high density of international obligations, only the international community could adjust the term of protection

1444 See N 562.

1445 See N 563; see similar reasoning in SAMUELSON (2017b), 1514.

1446 See N 253 for UCC, N 250 for RBC and N 259 f. for TRIPS Agreement. It is noteworthy that Art. IV para. 2 UCC already allows a reducing of the term of protection to 25 years. Due to conflicting provisions in other treaties, however, it is of practically no significance. Regarding the EC Copyright Term of Protection Directive, see also VON LEWINSKI SILKE.

for computer programs. The individual jurisdictions are bound by these international treaties, and could thus not abandon the current regulation of the protection term in their legislation. In light of the short and still shortening software life cycles, it would nevertheless be reasonable to find a more adequate statutory solution than the 20 years of protection after application in patent law and 50 years after the author's death in copyright. I hence recommend a regulatory change of the current protection terms for computer programs in copyright and patent law.

2. Connection Point

822 The international treaties govern that a work's term of protection in general ends 50 years after an author's death in copyright and 20 years after the application date in patent law.[1447] Art. IV para. 2 UCC and Art. 12 TRIPS Agreement further allow the legislators to tie the maximum term of protection in copyright to a connecting factor other than the life of a natural person. However, in all jurisdictions examined for this thesis – Switzerland, the United States and the observed member states of the European Union, a work's term of protection in copyright is currently tied to the death of the last living author.[1448] It is person-centred. While the point of reference in patent law is closely connected to the invention and its first full disclosure, in copyright the point of reference does not consider in any way the work's date of creation or release.

823 The interviewees stated that tying the expiration date of an intellectual property right to the developer or engineer *ad personam* would not be reasonable where created goods showed more economic or technical relevance than artistic. Most of the software companies therefore believed that the end of the protection term of a computer product should be tied to the computer program itself.[1449]

824 This argument is supported by the fact that nowadays most computer programs are developed in corporate structures, mostly by a team of representatives of different departments.[1450] Individual and independent software engineers publishing software are still common but seem to be the minority case.

1447 Art. IV para. 2 UCC, Art. 7 para. 1 RBC and Art. 12 and 33 TRIPS Agreement; see also N 253 for UCC, N 250 for RBC and N 259 f. for the TRIPS Agreement; for patent law in particular see N 336 ff., for copyright in particular see N 369.

1448 See N 259 f. for TRIPS Agreement and above N 369.

1449 See N 499.

1450 See same conclusion in THOMANN (1998), 20.

This same understanding has been incorporated into the software copyright law of most countries, usually assigning the rights to computer programs directly to the employer of a software developer instead of the author.[1451] The product and its development remain the key element in focus. However, in today's practice software is constantly being added, altered or deleted, and the subject matter of protection is hence continuously being changed. Consequently, to reconcile the term of protection against the specific object of protection, connecting the term of protection to the author would entail that a company would need to record for each (part of a) software project those participated in its development and when the last co-author died in order to determine the term of protection for the specific subject matter still in place at the time. If the individual contributions could be separated, even the individual work parts (e.g. components) would have to be attributed. According to the feedback of the software companies, this is nearly impossible.[1452] Tying the term of protection to the author's life expectancy thus contradicts the practicalities in software engineering and does not meet the needs of the software companies.

825 It would therefore be reasonable to tie a computer program's term of protection more closely to the creation in question. This could, for example, be conducted by tying the term of protection to an objectified connecting point, such as the work's *creation*[1453] *or when it was made available to third parties*, e.g. through its release. Although this would require a legislatory revision of many copyright laws, it would not represent an entirely new modification that is alien to the copyright system. Already today, individual copyright laws, such as the Swiss Copyright Act in Art. 31 sect. 1, provide that in the case of unknown authorship a work's expiration shall be linked to the publication of the work. The RBC also provides in Art. 7 for various scenarios in which a linking of the

1451 See Art. 17 Swiss CopA and § 69b Abs. 1 German UrhG. See also discussion in N 499.

1452 See N 499; see also decision of the Court of Justice of Geneva of August 6, 1986, published in SMI, 1987, 217 ff.; DESSEMONTET, N 984, with reference to Art. 30 para. 1 Swiss CopA.

1453 Following the model suggested in N 785 ff. above, the term of protection should hence in *linear development* start shortly after the product reaches its creative threshold. In case of *spiral development*, this would generally be when a sophisticated prototype of the software was achieved. Further thought would have to be given to *incremental developments*, as in continuous delivery, since in this case further modules are later added to the originally released group of modules. It is unclear when the term of protection for the later parts should start to run in this scenario. Inner changes on the other hand would in general not have an effect on the copyright's expiration.

expiration period to another event is legally foreseen. Even if the presented change involved a certain revision effort, it would in principle be systemically possible and useful in the result.

C. Conclusion

826 This chapter outlined how the term of protection is determined in the current legal models in copyright and patent law. The interview series brought up some difficulties of this approach for practitioners. It was therefore discussed where the starting point for the term of protection could be set appropriately relative to the particular development model in question. The work in question would need to be examined individually in order to determine the creative threshold. It was further suggested that the starting point for IP protection should be set earlier in the development process so that the project would be better protected from third-party interventions.

827 Considering the end point of the protection term, it was explained that the current model shows two major difficulties. First, the granted protection is too long compared with the estimated product life cycles of software products. It was therefore suggested to shorten the duration of patent and copyright protection to 10 and 20 years respectively. Second, tying the term of protection to the author's death in copyright seems inappropriate for a good that shows mainly economic and technical relevance rather than artistic. It was recommended to link the term of protection to the work's creation or its release instead of to the developers as work authors.

V. Excursus: Second-Hand and Dependent Creations

828 The following section is, according to the definition of the research object, not in the focus of this thesis. However, I would still like to include it, for two reasons: The first is that the question of dealing with dependent works is closely linked to the protection scope of creations in copyright and patent law as it can affect a work's or invention's eligibility for patent law and copyright protection. A strict separation of these topics would therefore not be appropriate. Second, the interviews provided relevant insights to these questions, which are worth sharing. For this reason, the following is designed as an excursus on second-hand works and dependent creations.

829 In the interview series, new technical approaches were described as providing new possibilities to effectively copy and paste single pieces or even whole

components of existing software products.[1454] At the same time, the increased availability of online communities and data bases has made it easier to access know-how in the form of technical solutions for reuse and integration into software products.[1455] Scotchmer describes how computer science works as "*dwarfs standing on the shoulders of giants*".[1456] She hereby refers to a famous depiction first explained by Bernhard of Chartres. It implies that in science new findings of a particular generation are often rooted in findings made by earlier generations.[1457] The knowledge of how to develop creations thus often builds on previous insights. In software development, adopting the design and structure elements of somebody else's work represents a widely established practice.[1458] In the ICT communities, sharing technical solutions was described as prestigious,[1459] and stealing as recognition of an original work.[1460] However, this does not mean that every new invention in computer science represents a copyright or patent law infringement. Rather, it tries to explain that new ideas often rely on established knowledge – the earlier findings of 'giants'.

830 In practice, the software components of third parties are not usually integrated into a software product one-to-one, but instead are adapted to the setting of the desired technical environment.[1461] From an ICT economic perspective, recycling old solutions is often more efficient because the software engineers do not have to reinvent the wheel for every new project but instead are able to build on the expertise of other engineers and prior works.[1462] But from a legal perspective, if developers use a third-party piece of work to realize their own product, either the consent of the right holder or a statutory exception is required. The essential question is how and to what degree can law include the circumstance that in a particular field of science building on pre-

1454 See N 401 ff., N 404 in particular.

1455 See N 398 f., N 421 ff., N 448 ff., N 489 f. and N 526; see also Fisher, 18 f.

1456 Scotchmer (1991); see also later perception in Scotchmer (2006), 127 ff.

1457 First cited in a letter from John Salisbury in 1159, printed in McGarry, 167; cited later by Sir Isaac Newton: "If I can see further than anyone else, it is only because I am standing on the shoulders of giants"; mentioned in transcript A, N 55 f.

1458 See N 398 f., N 448 ff., N 458 ff. and N 567 ff.; also discussed in: Samuelson et al., 2329 f.; Fisher, 18; U.S. Congress (1990), 12 and 15; Krueger, 131; see also general discussion in Landes/Posner (2003), 4; Scotchmer (2006), 129 f.

1459 See N 204 ff. and N 448 ff.

1460 See N 448 ff. and transcript K, N 53.

1461 See N 448 ff. and N 458 ff.

1462 See N 459; also described in Krueger, 133; see also Fisher who outlines a scenario in which the combination of various ideas may be economically more valuable than "the sum of its parts" (Fisher, 18).

vious findings and integrating them into new solutions constitutes part of the working culture and is also crucial for technological advances. This issue is closely connected to the question to what degree secondary works are allowed to build on existing ones. In the interview series, two people discussed the issue of derivative works with the help of the "chair dilemma": "Would you like it if a green chair and a yellow version of the same chair were each allowed to have an independent copyright?"[1463] After all, "a chair is a chair!"[1464] In my understanding, this dilemma involves two main problems: First, we might ask what law would require in order for the yellow chair to become independent so it does not fall into the green chair's copyright scope? Second, to what extent should the author of the yellow chair be allowed to build on the disclosed know-how of the green chair's builder on how to construct a chair, in case the outlined approach was considered ground-breaking and would, in future, be the only true way of chair construction?

Based on these problem spheres, the following discussion is split into two parts, both addressing the question of where the borders of the scope of protection lie with regard to derivative creations. The first one covers the question to what extent the right holder should be allowed to forbid subsequent works. As this question is particularly disputed in copyright, the following section focuses on this perception. The second part looks at the extent to which dependent creations have positive effects in the software industry and how they could be encouraged with legal measures. As the bar for an IP-admissible derivative is higher in patent law, this section focuses on the patent perspective of this problem. 831

A. The Right Holder's Right to Protect a Work from Further Processing

With the rise of newer development trends and the public offering of ready-made components on specialist platforms, in libraries and within the Open Source Community, it is important to assess to what extent computer programs are protected from being edited by other parties and in what sense right holders can forbid others from processing their software products. 832

1463 Transcript F, N 115.

1464 Transcript E, N 155.

1. Edited and Derivative Works

833 While patent law has a very broad protection scope and prohibits processing and imitating the patented subject, independently of the exact method of realization,[1465] copyright focuses on the particular expression of creative inspiration.[1466] In copyright, the scope is thus much narrower. It is difficult to determine whether an amendment can still be prohibited by the right holder or whether an independent new work outside his/her right of exclusion exists. The interviews revealed that it is extremely hard for software companies to evaluate whether or not a third-party use represents an infringement.[1467] This is particularly true for copyright, because the delimitations of permitted and inadmissible third-party uses are more a matter for discretion.

834 The interviewed software developers emphasized that being inspired by other software solutions and partially taking from other engineers should not be completely prohibited, as it was, firstly, widely established in the culture of the software industry, and secondly, necessary to realize a creative process efficiently. They outlined that there would always be certain similarities between two software products. The decisive question should be how much similarity two products are allowed to exhibit and what the author of the later work has contributed to create the the latter. *We are therefore talking about the quantitative similarity of two works as well as the quality and strength of the original and its successor*.[1468] At the same time, the software companies were insistent that a derivative work should exhibit a creative individual contribution in order to be legally recognized.[1469] Slavish copying was described as inherently uncreative and not worthy of copyright protection. It was summed up by the term '*copy with pride*', meaning that the copier or imitator also has to invest effort.[1470]

835 It is undisputed that the right holder of an intellectual property has the right to decide over their good and to protect it from being distorted and changed, if a third party tries to do so without their consent. According to the internationally codified understanding of copyright law, the one-to-one copying of computer programs is explicitly prohibited. The author has a broad set of

1465 See above N 268.
1466 See above N 348.
1467 See N 590 ff., N 591 in particular; see also the legal perspective in Straub (2001a), 810.
1468 See discussions in N 448 ff., N 458 ff., and 567 ff.
1469 See N 454 ff., and N 568 ff.
1470 See N 537.

rights to decide how their work will be utilized and distributed. However, the author's rights are legally constrained through certain statutory limitations and exemptions.[1471] A derivative work is regarded as an independent work in copyright if it exhibits enough originality of its own.[1472] In *Whelan v. Jaslow*[1473], the court determined that making small changes to somebody else's computer program would, however, not legitimate the use of somebody else's work without their authorization. Instead, the secondary work has to be *distant* enough from the original work to not fall within the protected scope of the original. Also according to the Swiss interpretation, the work would otherwise not be statistically unique and thus would not be original. It was mentioned before that although the European and U.S. practices have created their own judicial approaches to assess second-hand works, all the methods are similar in result:[1474] The more original and (statistically) unique the original work is, the greater its creative strength and protection and the less visibly dependent or similar a secondary work is allowed to be. The required distance of the secondary work from the original can only be achieved if the derivative exhibits its own *originality* and exhibits few substantial similarities. Therefore, careful consideration must be made as to whether the elements adopted by the second-hand work are of subordinate importance in view of its originality.

I believe that the rules the European and U.S. practice have developed over the years for classic copyright can be analogously applied to software. It meets the demands of the software companies that not every use of a copyrighted work should be forbidden. Instead, it is a matter of discretion whether or not a second-hand work still falls within the protective scope of an original, if the secondary work is too similar, and whether it offers enough quality to be deemed a copyrightable work itself by offering its own personal contribution.[1475] A right holder can only forbid the use of their work to the extent that the third-party work is remarkably close in the vital parts of their work, and if the new work does not offer enough personal creative involvement to be regarded as an original work. However, this discretionary decision as to whether a secondary work is dependent or original is very difficult to make in practice, and compa- 836

1471 See N 234 ff. for the function of intellectual property law, and N 342 and N 365 f. for limitations and legal boundaries in copyright.

1472 See N 365 f.

1473 *Whelan Associates v. Jaslow Dental Laboratory*, 797 F.2d 1222 (3d Cir. 1986); see also *Bridgeman Art Library, Ltd. v. Corel Corporation*, 36 F. Supp. 2d 191 (Southern District of New York 1999).

1474 See N 365 f.

1475 See N 454 ff. and N 568 ff.

nies can only partially anticipate infringements. Even though the U.S. jurisprudence has developed several different approaches to implement a substantial similarity doctrine, a general and universally applicable interpretation of a rule to compare two creations is still difficult to find. Unfortunately, this thesis is also unable to offer a definitive rule for this problem as this question does not fall directly into the scope of the thesis.[1476] It remains necessary to decide for each case separately whether a work shows a relevant similarity or originality of its own. Based on the results of the interview study, we can at least say that the existing court practice in principle incorporates the right elements.[1477]

2. On Translations in Particular

837 Another problem associated with derivative works and adaptations in computer programs are *translations* of a program's source code from one programming language into another. Particularly delicate is the question to what extent these translations are covered under an author's right to forbid editing of his or her works. While the EU Computer Program Directive in Art. 4 explicitly names the translation of computer programs as a copyright infringement, the U.S. and Swiss law do not address the issue of translation for computer programs separately. On the other hand, the interview study has shown how easily software products can be adapted, reformulated and applied to a different technical environment. The software companies emphasized that the easiest way to imitate a software product is to translate the source code from one programming language into another one of the same language family.[1478]

838 For classic literary works, several international treaties and national statutes explicitly categorize the translation of these as a special statutory form of editing that is subject to the author's approval.[1479] Again, this is true up to the point when the derivative work is able to show enough originality so that it represents an independent work of its own. The problem is that this protection was originally designed for translations of texts from one linguistic human language into another one. *It is not beyond doubt that translations of a source code*

1476 See interesting discussion in McGahn, although slightly out of date now.

1477 On the basis of the interview results, one could develop a model to compare software components in order to evaluate infringements from a copyright and patent law point of view. However, this would have to be done outside of this doctoral thesis, as a separate project.

1478 See N 401 ff. and N 568 ff.

1479 See for example Art. 2 para. 3 and Art. 8 RBC, Art. V UCC, Art. 3 para. 2 Swiss CopA, § 3 German UrhG and 17 U.S. Code § 101.

are similarly interpreted and protected against translation. This problem, although partially discussed in the literature, has not yet been resolved with clear court practice. In the United States, the Court of Appeals for the 3rd Circuit in *Whelan v. Jaslow*[1480] assessed a responsible first-instance trial court's decision as "not clearly erroneous", and preserved the court's decision to subsume source code translations as potential infringements of a work. However, the very reticent final remarks of the court left it unclear as to whether their decision in 1986 was supposed to represent a precedent case, or whether the grounds for appeal did not suffice to overturn the first-instance court's ruling. In Europe, to my knowledge, there is no precedent of a higher court on this question. In Swiss doctrine, reformatting, for example, is not considered as a prohibited interference or unauthorized modification in the sense of copyright law.[1481] In the additional literature it is intensively debated to what extent translations of the source code are copyright relevant. Some authors argue that if a translation tool was used, the final result, although expressed in a different form of mechanical realization, would include the original's creative achievement. All differences from the original would only result from the translation algorithm, which did not entail a creative contribution of its own.[1482] Others are critical and instead appear to classify computer translations as classical adaptations. Accordingly, they focus on the elements involved in the derivative and state that transformations into other forms of expression are not covered under the author's rights.[1483]

As Wittmer describes, the potential variety lies in the formulation of a program, its structure and its definition.[1484] One could therefore assume that, provided the decisive parameters and structures are still contained in the translated work, the processing would be forbidden because the derivative falls into the original's scope of protection. But according to the software companies, in software engineering there are more ways to express the same idea or func- 839

1480 *Whelan Associates v. Jaslow Dental Laboratory*, 797 F.2d 1222 (3d Cir. 1986); see also *Bridgeman Art Library, Ltd. v. Corel Corporation*, 36 F. Supp. 2d 191 (Southern District of New York 1999).

1481 See discussion in Staempfli Commentary to the Swiss CopA (Pfortmueller), Art. 11, N 4, with reference to Barrelet/Egloff, Art. 9 N 6a.

1482 See above N 574; see also discussion in Straub (2001a), 831 fn. 112; Kummer, 203 f.; Marly, N 174 and 177; Mooers, 60; Harison, 183; Neff/Arn, 158; Zirn, 39 f. ; Koehler, 88 f.; Samuelson (2012), 159; Rauber (1992), 43.

1483 See discussion in Staempfli Commentary to the Swiss CopA (Pfortmueller), Art. 11 N 4, with reference to Barrelet/Egloff, Art. 9 N 6a.

1484 See Wittmer, 106 f.; Samuelson et al., 2317.

tionality than in other natural sciences. The difference and creative distance from the original software depends on the effort and the resources somebody wants to invest.[1485] At the same time, the translation often visually differs significantly from the original, because another programming language was utilized.[1486] Elements in the original are therefore *no longer recognizable* in the translation, although it still follows the same elaborated commands. Under the existing practice to assess secondary works, program translations would not be legally qualified as an undue modification, even though the person copying the source code and entering it into a translator would not offer any personal contribution to the outcome.[1487] The assumption that computer translations are regular edits, as suggested by a part of the doctrine, could therefore *endanger the value of software* – its content and the included know-how. Treating computer languages differently from human languages could lead to *unfounded discrimination of digital literary expressions*.[1488]

840 I agree with the software companies that there are no grounds to differentiate between a linguistic translation and an imitation of a literary command in the source code. According to the precedent *Signo Trading International v. Gordon*[1489], in order to legally evaluate translations, the individual *personal contribution* the translator offers in his or her work is decisive for its copyrightability. In the case of human languages, one-to-one translations of single words and short phrases do not represent a creative contribution that is worthy of legal protection. Instead, an achievement beyond the "fairly mechanical process" is required. If no such contribution is exhibited, the result is achieved solely through purely technical or functional control elements of a translation software, which, on its own, does not involve any human influence and consequently would not result in an *intellectual creation of the human mind* from the perspective of copyright. There would be no valid second-hand work to legally assess. Further, in order to evaluate derivatives in copyright, we have to determine whether they exhibit *adequate originality* of their own. The fairly mechanical process of translation – the mere craft of translating a code line from one programming language into another – does not suffice for this matter, as only the rules of translation are applied, with no further achievement offered by the translator. A different interpretation of source code translations would

1485 See N 454 ff. and N 549 ff.; see the same opinion in: HARISON, 188 ff.

1486 See the same reasoning in PERELMAN, 944 f.

1487 See above, N 574, with further remarks.

1488 See N 573 ff.

1489 *Signo Trading International, Ltd. v. Gordon*, 535 F. Supp. 362 (Northern District of California 1981).

also be indefensible from the perspective that in classic literature the characters and plot of a work are also partially protected against imitation and copying in different work categories.[1490] In the case of computer programs, as long as a language of the same programming language family is used, the main composition remains, likewise, substantially unchanged.[1491] If the translator does not offer any additional personal contribution apart from the "craft of literary translation", the translation is too similar to the original and thus should require the consent of the original's author.

In that regard, the general rules the U.S. court developed in *Signo Trading International v. Gordon*[1492] can be directly applied to translations of other types of literary expression, such as translations between programming languages. But I would further propose that the protection of translations is extended and explicitly incorporated to computer programs and programming languages in the national statutories and international copyright agreements. From the described legal perspective, incorporating the legal protection of source code translations would be stringent. 841

B. The Community's Right to Profit from an Exclusive Property

In the last section, the right holder's right to control the utilization and distribution of his or her creation was discussed. In exchange for this exclusionary right, society and the market are entitled to insight and access to the protected findings.[1493] Where the balance between the interests of the right holders and society is lost, and the exclusivity granted to patent holders goes beyond the necessary level, "it has a disturbing effect on competition."[1494] The interviews showed how important competition is for the software industry.[1495] The law tries to account for this and sets certain rules within the framework of antitrust law and with the help of limitations in IP law. At the same time, the software market exhibits a particularly strong exchange culture, which may conflict with the author's right to control utilization and distribution of his or 842

1490 See the arguments in: decision of the BGH of July 17, 2013, I ZR 52/12 – *Pippi Langstrumpf*, published in GRUR, 2014, 258 ff.

1491 See N 401 ff., N 568 ff. and N 571 in particular.

1492 *Signo Trading International, Ltd. v. Gordon*, 535 F. Supp. 362 (Northern District of California 1981).

1493 See N 234 ff.

1494 Hilty (2018), 1186; see also Arrow, 226 f.; Cohen/Lemley, 39 and 50 ff.

1495 See N 411 ff.; see the same reasoning in Lemley/Burk, 90.

her work. Krueger describes how the reuse of software components can be inspiring for software engineering.[1496] It helps to spread knowledge. Likewise, the interviewed software companies emphasized that learning from other developers and exchanging know-how is important for the software industry.[1497] They explained that many software components are built on previous discoveries.[1498] Like software developers "sitting on the shoulders of giants",[1499] technical evolution as well as the quality of developments was said to be based on a *quality ladder*;[1500] which is path dependent. In this context, Cohen and Lemley stress how important it is to grant people other than the inventor access to developments so that these third parties can improve the existing inventions and state of the art.[1501] They propose that knowledge obtained and disclosed within the scope of intellectual property law should be made available for dependent inventors to develop improvements and introduce the new findings to the market.[1502] Similarly, Samuelson et al. state that the law should not completely forbid the derivative use of software components, and the creation of dependent works, but instead should try to regulate how fast and under what conditions dependent works can be introduced.[1503] The suggested system is therefore not a question of legal limitations and exemptions, but rather of determining the outer borders of the scope in patent law.

843 The following section deals in more detail with the significance of path dependency in software development for two specific scenarios. It discusses the extent to which patent law itself could provide rules to address the issue of dependent inventions in the form of compulsory licences for standard essential patents and incremental improvements. The legal limitations in patent law will not be further addressed.

1. Standard-Essential Developments

844 The software companies explained that there were certain inventions covered by absolute and exclusive rights that represented essential software components for the software industry, such as *technical necessities* or *engineering*

1496 Krueger, 131 ff.
1497 See N 448 ff.
1498 See N 458 ff. and N 567 ff.
1499 See Scotchmer (1991), 29 ff.; Scotchmer (2006), 127 ff.
1500 Scotchmer (2006), 133.
1501 Cohen/Lemley, 23 ff.
1502 Cohen/Lemley, 37 f.
1503 Samuelson et al., 2380 f.

standards. The technical solution contained in these components represents the most efficient and reasonable way of tackling a problem, or it may even represent the one available technical option. Although the developers explained that they can partially bypass a proprietory technical solution and find an alternative, they emphasized that there were certain technologies that they could simply not forgo.[1504] This problem is compounded by the fact that standardized processes and tools are very common in software engineering, and sometimes they underlie a patent. If standards are monopolized with an exclusive property right, other market participants are severely restricted, possibly even completely impeded during software development and commercialization. According to the interviewed companies, in the area of software engineering, the problem of patented basic inventions currently presents itself particularly in designing web shops and in the development of smartphones with suitable operating systems. Lemley explains the issue in simple terms: "Telephones talk to each other, the Internet works, and hairdryers plug into electrical sockets because private groups have set 'interface' standards, allowing compatability between products made by different manufacurers."[1505] The rules governing IP would therefore have a significant impact on standardization.[1506] The interview study showed that it would be important for software developers that standard-essential patents and other protected knowledge of industrial standards became better accessible on downstream markets, and could be used through a facilitated mechanism. Further, the interviewed companies highlighted that, as the computer science is still rather young, advancing it is still significant for science itself as well as for the society that should profit from it. Hence, some of the software companies suggested that right holders of a standard essential patent should be forced to grant access to their invention, for example through compulsory licensing. It will be discussed below how antitrust law has already made efforts to address this problem around standard essential inventions, either by indirect restraint to license or by order to act. In the following, it is discussed to what extent it would be appropriate to provide specific mechanisms in patent law to limit the claims of patent holders which are likewise rooted in patent law.

1504 See N 402 f., N 512 ff., N 570 f. and N 577 ff.; see also discussion in KLEMPERER, 115 and 127; HEINEMANN (2002), 171 f. and 519 ff.; SCHWARZ/KRUSPIG, 43 f.; LEMLEY (2002), 1891 and 1895.

1505 LEMLEY (2002), 1893.

1506 LEMLEY (2002), 1895.

a) Types of Standards

845 Before engaging in a legal debate, I would first like to take a closer look at the concept of standards. As Bodewig explains, practices often rely on *industry standards*, that is, technical processes or systems that are uniformly applied in certain product or service markets. They are specifically designed to eliminate or reduce incompatibilities between different systems and vendors in order to ensure similar principles or simply to make the best technology available to the market.[1507] Standardization processes build on the idea that the whole industry should agree on one common procedure.[1508] Standards can be set formally by a certain body, usually an association or standard-setting organization for a specific national, regional or international territory, known as *organizational standards*.[1509] Standards can also develop on the basis of actual circumstances prevailing in a certain market.[1510] "If a product or process is considered by market participants to be the best or even without alternative and is (almost) exclusively used over time, a *de facto standard* is established".[1511] Both organizational and de facto standards are present and well-established in software science. Where these standards, cumulative innovation, and (multiple) blocking patents meet, the patents may start impeding innovation. Shapiro in this context speaks of a *patent thicket*, "a dense web of overlapping intellectual property rights that a company must hack its way through in order to actually commercialize new technology."[1512] This thicket can quickly lead to a predicament. As Shapiro explains, in a situation where manufacturers are likely to stand on a land mine, they either have to "lose their corporate legs" being forced to pay disproportionate royalties on patents they could have invented around, or avoid the minefield altogether, meaning they will refrain

1507 Bodewig, 626 f.; Katz/Shapiro, 109 f.; Sommerville, 706 ff.; Bauknecht/Zehnder, 273; Wiebe, 23 and 439 ff.; Heinemann (2002), 520 f. Schwarz/Kruspig, 43 f.; Ullrich/Heinemann, N 37a.

1508 See also above, N 736.

1509 Hilty/Slowinski, 782; Lemley (2002), 1899.

1510 See the European Committee for Standardization's "Principles and Guidance for Licensing Standard Essential Patents in 5G and the Internet of Things (IoT), including the Industrial Internet" or the European Telecommunications Standards Institute's "Long Term Evolution Standards".

1511 Hilty/Slowinski, 782; see also Lemley (2002), 1899.

1512 Shapiro, 120; see similar discussion in Lemley (2017), 908 f. and 928 f.; Wiebe, 439 ff.

from introducing products in this industrial field due to a fear of *holdup*.[1513] The limited or lacking access to patented standard essential innovations may thus result in market failure.

b) The Antitrust Approach

Fortunately, the problem of possible market failure has not gone undetected. Jurisprudence as well as certain national and regional trade organizations have recognized this issue. The European Commission in the cases of *European Commission v. Samsung*[1514] and *European Commission v. Motorola*[1515] introduced the so-called *essential facilities doctrine* to software patent law which aims to enable the use of a protected fundamental development, if society can otherwise not obtain access.[1516] Legally, the European Commission based their intervention against the patent holder on a preliminary view, a statement of opinion before a formal ruling is issued, that both Motorola and Samsung seeking and enforcing an injunction against Apple in Germany on the basis of mo- 846

1513 SHAPIRO, 125 f.; see also LEMLEY/SHAPIRO, 1940, with further references; SCHWARZ/KRUSPIG, 44 f.; see also discussion in HEINEMANN (2002), 520 f., with a different terminology but discussing the same problem.

1514 European Commission, decision of April 29, 2014 (case no. 39939) – *Samsung*, involving organizational standards.

1515 European Commission, decision of April 29, 2014 (case no. 39985) – *Motorola*, involving organizational standards.

1516 The essential facilities doctrine had previously already been established for several other monopoly fields, such as de facto obstruction of access to certain facilities. See for examples of privileged access to essential inputs or natural resources, such as important technologies (e.g. decision of the Judgment of the Court of First Instance of December 12, 1991 [T-30/89] – *Hilti v. Commission*), an established distribution and sales network (e.g. decision of the ECJ of February 13, 1979 [C-85/76] – *Hoffmann-La Roche v Commission*), costs and other impediments, for instance resulting from network effects faced by customers switching to a new supplier (decision of the ECJ of February 14, 1978 [C-27/76] – *United Brands v. Commission*). It was further extended to intellectual property law where there was no actual or potential substitute on which competitors in the downstream market could rely so as to counter the negative consequences of the refusal on the dominant undertaking, for example copyright law (e.g. decision of the ECJ of April 6, 1995, [joined cases C-241/91 P and C-242/91] – *Radio Telefis Eireann [RTE] and Independent Television Publications LTD [ITP] v. Commission [Magill]*. See also discussion regarding essential facilities and antitrust law in HEINEMANN (2002), 502 ff., particularly 510 ff. and 521 f.; WOLFF, 157 ff. See also the Guidance Paper of the European Commission on Abusive Exclusionary Conduct by Dominant Undertakings regarding further information on their take on the refusal to deal and the essential facility doctrine.

bile phone standard essential patents amounted to an *abuse of a market dominant position prohibited by EU antitrust rules*. The European Commission noted in its press release on the Motorola case[1517] that while recourse to injunctions would be a possible remedy for patent infringements (as advocated by Samsung and Motorola), such conduct may be abusive where standard essential patents were concerned and the potential licensee would be willing to enter into a licence on *Fair, Reasonable and Non-Discriminatory terms* (so-called FRAND terms). The Commission considered that in such scenarios dominant patentees of standard essential innovations should not have recourse to injunctions involving prohibition to sell the product infringing the patent in order to distort licensing negotiations and impose unjustified licensing terms on patent licensees. Therefore, if Motorola did not grant licences to the other market participants, the competition authorities might judge this as a sanctionable act under competition law.[1518] The Commission hereby transferred its practice in settling the *Rambus*-case[1519] in the field of patented computer chips to software, and at the same time adopted the practice of the German Federal Court of Justice in 2009[1520] as well as the long-standing practice in the United States[1521], in which the monopoly-like position of patent holders in the field of software is countered by means of antitrust law. As, in the cases under review, Motorola and Samsung decided to cooperate after the release of the European Commission's preliminary assessment, and, thus, granted its competitors access to the inventions, no formal ruling was made.

1517 For the full press release of May 6, 2013, on the Motorola case, see <http://europa.eu/rapid/press-release_IP-13-406_en.htm> (retrieved September 6, 2021).

1518 European Commission, decision of April 29, 2014 (case no. 39985) – *Motorola*; see also application of antitrust law for the disclosure of interfaces and interoperability information in: decision of the ECJ of December 17, 2007 (T-201/04) – *Microsoft Corp. v Commission of the European Communities*, as well as the previous case in front of the European Commission: European Commission, decision of March 24, 2004 (case no. COMP/C-3/37.792) – *Microsoft*, comment in Thouvenin (2008b) and Heinemann (2005); see also general discussion regarding antitrust law and interfaces in Heinemann (2002), 514 ff.; *United States v. Microsoft Corp.*, 253 F.3d 34 (Columbia District Court Cir. 2001).

1519 European Commission, decision of December 9, 2009 (case no. COMP/38.636) – *Rambus*.

1520 Decision of the BGH of May 6, 2009, KZR 39/06 – *Orange-Book-Standards*, published in: GRUR, 2009, 694 ff., which involved a de facto standard.

1521 See for example *Associated Press v. United States*, 326 U.S. 1 (1945); *Lorain Journal Co. v. United States*, 342 U.S. 143 (1951); *Otter Tail Power Co. v. United States*, 410 U.S. 366 (1973); *RSA Data Security, Inc. v. Cylink Corp.*, Civ. no. 96-20094 SW., 1996 WL 107272 (Northern District Court of California 1996); see also Waller regarding new markets such as software and social media.

In *Huawei Technology Co. Ltd. v. ZTE Corp., ZTE Deutschland GmbH*,[1522] the European Court of Justice confirmed the Commission's argument that the abuse of patent rights, in particular the refusal of licensing, may in particular cases constitute misconduct under antitrust law. The dispute between Huawei, the Chinese mobile network technology company, and the Chinese company ZTE was about ZTE's marketing of mobile network products in Germany which involved a software patented link to the 4G Long Term Evolution standard. In 2009, Huawei made a commitment to the European Telecommunications Standards Institute, a standard-setting organization at European level, to license a standard-essential declared European patent it held to the Long Term Evolution standard to other parties on FRAND terms. In accordance with this engagement, the parties entered into negotations to conclude a licensing agreement for Huawei's patent, but did not succeed. Thereupon, Huawei filed an injunction at the Dusseldorf Regional Court against ZTE for the rendering of accounts, the recall of products, and an award of damages based on an alleged patent infringement. For clarification purposes, the Court referred a set of questions to the European Court of Justice, who ultimately held that the refusal to license under FRAND conditions may constitute a violation of antitrust law in two ways: Firstly, with the existence of a standard-essential patent the patentee could prevent competitors from manufacturing downstream products. On the other hand, the FRAND commitment created legitimate expectations on the part of third parties that the owner of the standard-essential patent would in fact grant licences on such terms. Even if the owner of a standard-essential patent, like every right holder, must have the possibility to take legal action against infringers, they are obliged to account for special circumstances, if they have given an irrevocable FRAND commitment. With regard to the latter point, the European Court of Justice identified a number of specific steps necessary to legitimate a right holder's application for an injunction so 847

[1522] Decision of the ECJ of July 16, 2015 (C-170/13) – *Huawei Technology Co. Ltd. v. ZTE Corp.*, ZTE Deutschland GmbH.

they would not fall under the provision against abuse of dominance.[1523] The Court further held that patented inventions that were included in the collection of a standard-setting organization should be considered as standard-essential patents, because they rendered its use indispensable to all competitors who envisaged manufacturing products that complied with the standard to which it was linked.[1524] In passing, the Court also addressed the circumstance that the German Orange-Book-case, on the other hand, contained a de facto standard that was not based on a FRAND declaration but on an established licensing program of the patent owner.[1525] Accordingly, both, the standards imposed by organisations as well as the de facto standards were determined as possible subjects of standard-essential patents to the extent that either a FRAND declaration or an established licensing program with additional commitments towards the seeking licensee were provided.

848 To complete the short tour through antitrust law, it is worth taking a glance at the United States. Here we can observe a two-part practice in dealing with standard-essential technologies. On the one hand, the U.S. Federal Trade Commission, like the European practice, regards the enforcement of a patent in an injunction action against a FRAND declaration towards a standard-set-

1523 Firstly, prior to bringing that action, the proprietor has to alert the alleged infringer of the infringement complained about by designating that patent and specifying the way in which it has been infringed. Secondly, the infringer must express their willingness to conclude a licensing agreement on FRAND terms. Thirdly, the proprietor has to present to that infringer a written offer for licensing, in which they specify the terms, in particular the royalty and the way in which it was calculated. Fourthly, the infringer has to respond diligently to that offer and in accordance with recognized commercial practices in the field as well as in good faith, which must be established on the basis of objective factors and which implies, in particular, that there are no delaying tactics. Further, if the infringer is using the teachings of the patent in dispute before a licensing agreement has been concluded, the implementer must provide appropriate security in respect of its past and future use. Finally, where the parties have not reached an agreement on FRAND terms following the infringer's counter-offer, they may request that an independent third party determines the scale of the royalty fees. As the further details of the ruling are not relevant for this thesis, I refer to the specialist literature for further information on the subject: Heinemann (2015); Mamane; Picht (2018a and b); Hilty/Slowinski; Banasevic; Koerber; Maume; Killick/Sakellariou; Petit.

1524 Decision of the ECJ of July 16, 2015 (C-170/13) – *Huawei Technology Co. Ltd. v. ZTE Corp., ZTE Deutschland GmbH*, c. 47.

1525 Decision of the BGH of May 6, 2009 (KZR 39/06) – *Orange-Book-Standards*, published in: GRUR, 2009, 694 ff.

ting organization as anti-competitive behaviour.[1526] In a more recent case of 2019, the Northern District Court of California was able to review the FTC's practice in the Qualcomm case,[1527] in which no formal FRAND declaration was made but instead Qualcomm had stopped licensing its de facto standard technology to competitors. The court came to the conclusion that Qualcomm's refusal to license its standard-essential invention to its competitors impeded the competitors' ability to sell modem chips externally or at all, promoted their market exit, and delayed their entry. Qualcomm's refusal to license rivals had further limited original equipment manufacturers' chip supply options, which had enabled Qualcomm's anti-competitive conduct to require unreasonably high royalty rates from the original equipment manufacturers. The court held, inter alia, that Qualcomm should make exhaustive standard-essential patent licences available to modem-chip suppliers on FRAND terms and had to submit to arbitral or judicial dispute resolution to determine such terms. Secondly, and contrary to the aforementioned European practice, the U.S. practice further provides any person who is dependent on a standard-essential patent with an actionable civil claim to enforce the standard-essential patentee's declaration to a standard-setting organization and guarantee that the patentee offers their invention on FRAND terms to third parties.[1528] Dogmatically, this is constructed on the basis of an (implied) third-party contract; the patent owner's refusal to license a third party may cause this potential licensee to sue the patentee as a third-party beneficiary, while the standard-setting organization's claim against the owner remains separate. The third party's position relative to the standard-setting organization allows them to seek FRAND terms in the capacity of the intended beneficiary of the patent

1526 See for example: Decision of the Federal Trade Commission in the matters of MOTOROLA MOBILITY LLC and GOOGLE INC. of July 23, 2013, docket no. C-4410, available at <https://www.ftc.gov/sites/default/files/documents/cases/2013/07/130724googlemotorolado.pdf> (retrieved June 16, 2019); decision of the Federal Trade Commission in the matters of *Robert Bosch GmbH* of April 23 2013, docket No. C-4377, available at <www.ftc.gov/sites/default/files/documents/cases/2013/04/130424robert-boschdo.pdf> (retrieved September 6, 2021). This practice is partly based on the 1995 and 2017 revised Antitrust Guidelines for Licensing Intellectual Property (see U.S. Department of Justice/Federal Trade Commmission).

1527 *Federal Trade Commission v. Qualcomm Incorporated*, case No. 17-CV-00220 (Northern District Court of California 2019).

1528 *Microsoft Corp. v. Motorola, Inc.*, 696 F.3d 872 (9th Cir. 2012); *TCL Communication Technology Holdings, Ltd. v. Telefonaktiebolaget LM Ericsson*, case no. 14-341, 2017 WL 6611635, Central District Court of California 2017).

owner and the agreement with the standard-setting organization.[1529] The third party can hereby force the patentee to 'voluntarily' submit to the licensing terms promised to a standard-setting organization, and, thus, to proffer licences consistent with the commitment made.[1530] FRAND licensing hereby becomes a legally enforceable civil obligation on which third parties can rely. In view of this U.S. case law, the solution to a civil claim which is enforceable by way of a lawsuit only recently found its way into English common law jurisdiction.[1531] In both cases, the courts held that by means of a fictitious contract in favour of third parties, the holders of standard-essential patents had made a binding contract with their FRAND commitment to the European Telecommunications Standards Institute, that third parties could equally invoke to achieve a licence agreement. American jurisprudence regarding actionable claims has therefore already spilled over into common law in Europe. The double-track approach described in common law, using antitrust law to educate patent holders and offering a civil claim based on contract law to third parties to obtain a licence on FRAND terms, as Hilty and Slowinski rightly point out, merges logically.[1532] The common law courts have hereby constructively expanded the portfolio of possible measures in cases of standard-essential patents.

849 As a preliminary conclusion, the European and U.S. competition authorities and competent courts have recognized the importance of securing access to standard-essential patents. In dealing with this problem, most of the arguments so far have been based on an antitrust licensing obligation, due to abuse of market power, in connection with FRAND declarations for organizational standards and, sporadically, also to an established licensing program in cases of de facto standards.[1533] Following the approach advocated in this thesis, the standing practice concerning standard-essential patents may be analogously applied to software components such as algorithms, source code elements and specific functions, some of these already having been established as organizational standards or presented as de facto standards. In its result, the antitrust course seems reasonable. However, the exclusionary effect of a standard-essential patent is not limited to a possible abuse of market power. As Heinemann emphasizes in a different context, an IP right would not necessarily

1529 See Ragavan/Murphy/Davé, 93 and 94; Lemley/Shapiro, 1136 ff.

1530 *Microsoft Corp. v. Motorola Inc.*, 696 F.3d 872 (9th Cir. 2012), 12109.

1531 *Unwired Planet International Ltd v Huawei Technologies Co Ltd*, decision of the EWCA of October 23, 2018 (EWCA Civ. 2344), published in RPC, 2018, 757 ff.; *Apple Retail UK Ltd. v. Qualcomm (UK) Ltd.*, decision of the EWHC of May 22, 2018 (EWCA Pat. 1188).

1532 Hilty/Slowinski, 786.

1533 Heinemann (2015), 857; Waller, 1772 f.

cause a dominant market position. Rather, this case would be quite rare, which is why intellectual property law is not subject to systematic control on the basis of the prohibition of abuse under antitrust law.[1534] To resolve the problem of standard-essential patents, the abuse of a dominant position might hence not represent a sound anchor point. Rather, it is inherent that intellectual property rights, especially patents, as legally created exclusionary rights, can affect other market participants – neighbouring markets or the same one – and that these impacts may have an even more severe effect in the case of standard-essential inventions.[1535] Since the unique market position represents the intended outcome of monopolizing patent rights, in my opinion – and contrary to the competition commissions and courts – the negative effects patent regulation may entail should likewise be solved by patent law, and not with a detour around antitrust law.[1536] The great advantage of this approach from a legal point of view would be that one would not have to carry out complicated market delineations and prove that new and similar technical developments were prevented by an abusive act.[1537] instead, a systemic path would be followed.

c) *The Patent Law Approach*

Similar to what the interviewed software companies proposed,[1538] I would like to suggest a model of compulsory licences for standard essential developments in patent law. The mechanism of compulsory licences was introduced to achieve a balance between the patentee's right and the societal need for a product or process. If a patent holder is unwilling to license their invention, compulsory licences substitute a licensing agreement between them and a po- 850

1534 Heinemann (2006), 706; see also a statement of the European Commission, saying that legal monopoly-like rights do not necessarily have to result in predatory conduct (Guidance Paper of the European Commission on Abusive Exclusionary Conduct by Dominant Undertakings, fn. 2).

1535 Similarly, Hilty and Slowinski conclude that "the disadvantage of recourse to standards outside patent law and in particular to antitrust regulations lies primarily in the fact that these cannot adequately address the specific problem of [standard essential patents] in their general form" (Hilty/Slowinski, 786).

1536 See similar thoughts in Hilty (2018), 1190 f.; Samuelson et al., 2411; Merges/Nelson, 840, with further references; Harison, 44 f. and 116; Hilty/Slowinski, 786; see also discussion in Ullrich (1996), 565 ff.

1537 Heinemann criticized this aspect in the praxis of the European Commission (see Heinemann [2005], 74 f.; Ullrich/Heinemann, N 58 ff. and 61).

1538 See N 577 ff.

tential licensee in the form of an authoritative decision.[1539] The idea of applying compulsory licences to patent law is not far-fetched. Already today, the heavily codified international patent law provides certain rules according to which patent holders can be obliged to grant third parties rights to use.[1540]

aa) Based on a Superior Public Interest

851 Such interference in the patent holder's exclusive rights is in general only justified if there is a *superior public interest* in it. The term 'superior public interest' represents an indeterminate legal term and must be specified for each individual case.[1541] We can, however, only speak of a public interest if the need for use has a positive effect on the general public or the market and does not solely serve the individual dependent inventor who wants to use a patented invention.[1542] Again, we may adopt some learnings from the practice in antitrust law, where the European Commission argued in *IMS Health v. NDC Health* that in order to install a balance between the interests of an IP owner and free competition, the "latter should be given precedence over the interests of the IP owner where refusal to grant a licence prevents the development of the secondary market to the detriment of consumers".[1543] The interests of 'a secondary market' may hence amount to relevant public interests. In general, the public interest may encompass technical, economic and socio-political as well as medical aspects.[1544] However, in practice, its application has been largely equated with public health or measures against risks threatening the latter. For this reason, compulsory licensing in a public interest, if applied at

1539 Ragavan/Murphy/Davé, 108, with further references; Wolff, 20 f.; Bornhauser, N 259 f.

1540 See Art. 8 in conjunction with Art. 31 and revised Art. 31bis TRIPS Agreement. See also national provisions in Art. 32, 36 and 40 ff. Swiss PatG, § 24 and 85 f. German PatG, Art. L613-12 ff., Art. L613-15, Art. L613-17 ff. and Art. L613-19 f., Art. L623-17 French IP Code. In contrast to other patent statutes, the U.S. patent code itself does not include a general compulsory licensing provision, but instead restricts their use to particular domestic statutes on individual thematic areas, such as plant variety protection and atomic energy.

1541 Decision of the Swiss BPatGer of June 7, 2012, case no. O2012_021, c. 15; decision of the BGH of June 3, 1970, X ZB 10/70 – *Cafilon*, published in GRUR, 1972, 471 ff.; see also Wolff, 26.

1542 Troller (1985), 856; Wolff, 27.

1543 Decision of the ECJ of April 29, 2004, C-418/01 – *IMS Health/NDC Health*, c. 48.

1544 Decision of the BGH of July 13, 2004, KZR 40/02 – *Standard-Spundfass*, published in GRUR, 2004, 966 ff., c. III.1; decision of the German Patent Court of September 6, 2018, 3 LIiQ 1/18 (EP); see also Wolff, 29.

all, has been practically limited to ensuring worldwide availability of life-supporting drugs, such as AIDS therapies, and addressing hazards to public safety with the help of environmental protection legislation.[1545]

The applied scope of interpretation of the term superior public interest has thus been rather narrow in the jurisdictions examined.[1546] In this context, with regard to the latest developments and today's innovation market, I expect that in the near future a further significant increase in the number of software innovations in the healthcare and public safety sectors is going to occur. The Swiss State Secretariat for Education likewise emphasizes that Switzerland currently specializes above-average in health-related technologies, such as pharmaceuticals and medical technology.[1547] The significance of Switzerland in the future medical and biotechnology market is likely, considering that Switzerland is among the OECD's top R&D spenders in software.[1548] With more software inventions to come in this field, the governments and patent authorities will get another opportunity to assess the chances of compulsory licences in health-related patent law, as the societal interest in gaining access to disclosed know-how becomes ever more vital. With increased research activity and market development, the concerns of the market and society for uniform criteria and guidelines increases the need for consolidation in the form of standardization, which will become particularly important for the highly regulated field of public health and security. It should thus lie in the superior public interest for public health to foster standardization and to ensure this standardization with compulsory licences. 852

But public health and security should not represent the sole relevant factors in determining the public interest. In principle, we could also subsume *macroeconomic considerations* and consider the classic functionalities of IP rights.[1549] It would, for instance, be economically reasonable to facilitate access to an in- 853

1545 See for examples: for Switzerland Art. 40b Swiss PatG for biotechnological research tools, Art. 40c Swiss PatG regarding diagnostic tools for humans, and Art. 40d Swiss PatG for pharmaceutical export; for the European Union: Regulation (EC) No. 816/2006 of the European Parliament and of the Council of 17 May 2006 on compulsory licensing of patents relating to the manufacture of pharmaceutical products for export to countries with public health problems, Art. 13-17 of the Treaty establishing the European Atomic Energy Community; for the United States: the Clean Air Act, governmental march-in rights in the Bayh-Dole Act, the Food & Drug Act, the Public Health Price Protection Act.

1546 See the same perception in Wolff, 24.

1547 Swiss State Secretariat for Education, 90.

1548 Swiss State Secretariat for Education, 38.

1549 See also discussion in BGE 139 III 110, c. 2.3.3.

novation, if the exclusionary position of a patent holder influenced an industrial sector in such a way that market participants could no longer perform their work in a timely and demand-oriented manner. The goal of an intervening measure would thus be to allow market participants to provide their services efficiently by granting them access to basic technologies, and through this to reduce transaction costs. Particularly in the young software market, we can observe the need of users and downstream markets to use standard essential inventions at affordable and fair prices, as software developers either have to circumvent excessive hurdles or pay proportionally overpriced licence fees. Moreover, as already established by the English and Welsh Court of Appeal, the objectives of intellectual property rights involve society's public interest in profiting from an invention or work covered under exclusionary rights. "*As a society we want the best, most up to date technology to be incorporated into the latest standards and that will involve incorporating patented inventions. While the inventor must be entitled to a fair return for the use of their invention, in order for the standard to permit interoperability the inventor must not be able to prevent others from using the patented invention incorporated in the standard as long as implementers take an appropriate licence and pay a fair royalty. In this way a balance is struck, in the public interest, between the inventor and the implementers.*" [1550] From an intellectual property point of view, with this contemporary notion, compulsory licences would be permissible in the public interest if standardization significantly increased or ensured efficiency and meant that users (and dependent secondary markets) could proportionately participate in the provided advantages of an IP system. At the same time, the patent holder would be able to receive appropriate remuneration. This would represent a macroeconomic and also, from the perspective of IP law, balanced solution for all involved interests. It would therefore be appropriate to extend the interpretation of the concept of superior public interests as explained. A useful aspect of this extended definition would be that it could easily be integrated into the existing, heavily codified, patent law pursuant to Art. 8 in conjunction with Art. 31 and 31bis TRIPS Agreement, making a revision of national or regional legislation redundant. In addition, the existing regulations already provide for individual provisions on how compulsory licences should be implemented in detail.

[1550] *Unwired Planet International Ltd v Huawei Technologies Co Ltd*, decision of the EWCA of October 23, 2018 (EWCA Civ. 2344), published in RPC, 2018, c. 83; see also Arrow, 226 f.; Heinemann (2006), 705 f.

bb) To Mitigate Negative Effects of Patenting

But even if the current narrow scope of public interests were to be maintained, there is another, almost forgotten, legal basis in international law that allows compulsory licensing. Art. 5 lit. A Paris Convention states that the contracting states are free to implement a model that enables them to give users licences, *if the negative effects of patenting could hereby be mitigated.* In other words, countries are free to implement a system of compulsory licences in patent law if the monopoly-like patent rights show negative impacts of some kind. The potentially severe negative outflows of patents in the form of market obstacles to standard essential inventions in software engineering, as outlined and confirmed by the standing practice in antitrust law, involve relevant market impacts and may qualify as such 'negative effects of patenting' under the provision.[1551] Compulsory licences would hence represent a potential solution compliant with the international patent regulation under Art. 5 lit. A Paris Convention. The provision offers the national legislators a feasible regulatory mechanism that does not require a lot of effort for implementation. 854

cc) Notes on the Possible Licensing Procedure and Terms

I would like to propose a system in which software developers could apply for a right to use standard essential innovations at the locally competent patent authority or with which they obtain an actionable civil claim to achieve a licence to use a standard essential innovation. The subject matter, that we might consider as a standard-essential patent, could likewise be derived from the practice in antitrust law. In this context, it is important, as explained above, that on the one hand there are organizational standards set in a formal process by standardization organizations and accepted by market providers through a 855

[1551] However, they would not necessarily have to be combined with a dominant market position, as is currently the case under antitrust practice.

declaration of accountability,[1552] while, on the other hand, there are also de facto standards which develop from practical circumstances and either constitute the only available or the most expedient option.[1553] Both types of standards should be integrated into a potential system of compulsory licences. However, in the case of de facto standards, it should be further evaluated whether, as in the case of the FRAND practice, a firmly established licensing program of a patent holder or some other kind of contractual commitment should be required. Otherwise we would have to delimit when a patent holder who distributes their own patented innovations him-/herself could be obliged to grant a compulsory licence even without an externally affecting conduct.

856 If a statutory basis for compulsory licensing is set, the requirements and licence conditions should be regulated at the same time. Unfortunately, the Paris Convention does not provide any further information on the conditions for obtaining a compulsory licence or the conditions of use. There are, however, detailed provisions on the terms of compulsory licensing in the TRIPS Agreement and the the latter adopting national regulations that could be applied by analogy: Regarding the *procedure* to obtain a compulsory licence, Art. 31 lit. i and j TRIPS Agreement governs that both the granting of a right to use and the compensation to pay for it, are actionable. However, receiving a statutory compulsory licence from a court or authority is subsidiary to an independently reached contractual solution with the patent holder. Only if the patent holder within a reasonable period of time denies their authorization with market conditions, or cannot be found, can one apply for a governmental

1552 See N 845; organizational standards were accepted by the competition authorities in: European Commission, decision of April 29, 2014 (case no. 39939) – *Samsung*; European Commission, decision of April 29, 2014 (case no. 39985) – *Motorola*; decision of the ECJ of July 16, 2015 (C-170/13) – *Huawei Technology Co. Ltd. v. ZTE Corp., ZTE Deutschland GmbH* and decision of the Federal Trade Commission in the matters of *Motorola Mobility LLC* and *Google Inc.* of July 23, 2013, docket no. C-4410, available at <https://www.ftc.gov/sites/default/files/documents/cases/2013/07/130724googlemotorolado.pdf> (retrieved June 16, 2019); decision of the Federal Trade Commission in the matters of *Robert Bosch GmbH* of April 23, 2013, docket No. C-4377, available at <www.ftc.gov/sites/default/files/documents/cases/2013/04/130424robertboschdo.pdf> (retrieved September 6, 2021); *Unwired Planet International Ltd v Huawei Technologies Co Ltd*, decision of the EWCA of October 23, 2018 (EWCA Civ. 2344), published in RPC, 2018; *Apple Retail UK Ltd. v. Qualcomm (UK) Ltd.*, decision of the EWHC of May 22, 2018 (EWCA Pat. 1188).

1553 See N 845; de facto standards were accepted by the competition authorities in: Decision of the BGH of May 6, 2009, KZR 39/06 – *Orange-Book-Standards*, published in GRUR, 2009, 694 ff.; *Federal Trade Commission v. Qualcomm Incorporated*, case no. 17-CV-00220-LHK (Northern District Court of California 2019);

right to use.[1554] In cases of urgency, this step may be omitted. It certainly makes sense to prioritize individual agreements over government intervention. However, in return, a quick and efficient application or judicial process should be established for cases in which the patent holder refuses to negotiate, be it before a court of law or before a specialized authority. The ordinary judicial procedure in front of state courts in patent law is likely to take too long and not lead to the desired results in a timely and cost-effective manner. In the actual procedure, the competent office would probably first have to either establish the standard essential quality of a patent after a declaration by a standardization organization, or declare a patent to be a de facto standard. This decision again may either have an *inter partes* or an *erga omnes* effect for future rulings. The deciding authority would further have to determine for which *territorium* the compulsory licence was granted. According to Art. 31 lit. f TRIPS Agreement, authorities can predominantly authorize rights to use for the supply of a member state's domestic market, but only within their jurisdiction. In the past, courts have already considered extending the scope of statutory licences from national to regional ones, particularly with regard to the European Patent Organisation.[1555] We should also consider the *scope, duration and remuneration* of the licence. According to Art. 31 lit. a, c, d and e TRIPS Agreement, the exact scope of a compulsory licence should be determined on a case-by-case basis. The principle of proportionality plays a major role in this evaluation. In general, the licence shall be non-exclusive and non-assignable to third parties. At the discretion of the deciding body, a patent could, however, be assigned entirely to a third party or expropriated in the public interest, if particular welfare for society demanded it. The governing authority hence has a lot of discretion, especially when dealing with potential cases in the public interest. A right to use should be proportionate to the circumstances. As the European Commission clarified in its case against *Microsoft*, making available certain information regarding the specifications of a program does not amount to compulsory licensing of a source code.[1556] It seems the Commission with this statement wanted to circumscribe that where only a fragment or a specific part of a component is necessary to meet the needs of a

1554 See also discussion in HEINEMANN (2005), 495. If the patentee were to impose judicial prohibitions on the potential licensee after the latter had tried to obtain a licence, this would constitute an action relevant to antitrust law in accordance with previous practice on standard-essential patents.

1555 Decision of the German BPatG of June 7, 1991, 3 Li 1/90 – *Zwangslizenz*, 101 f.

1556 European Commission, decision of March 24, 2004 (case no. COMP/C-3/37.792) – *Microsoft*, c. 714.

licence-seeking party, the whole component or whole software should not be made available. Such an evaluation of proportionality requires not only legal but also technical judgement. As Heinemann specified, it depends on what is indispensable for carrying on a particular business.[1557] But with regard to the indisputable confidentiality interests of the right holder in the source code, a clear limitation of the compulsory licence to the absolutely necessary should be carried out on an individual case analysis and, where necessary, should be protected with additional measures such as trade secret protection for the compulsory licensee. Next, similarly to the practice in antitrust law,[1558] the TRIPS Agreement provides in Art. 31 lit. h that the right holder should be paid adequate *remuneration*, depending on the circumstances of each case, taking into account the economic value of the authorization. Like the FRAND terms under antitrust law, the compulsory licence would therefore be subject to a royalty fee. The use of the licensed subject matter is compensated, so the patent holder is not deprived of the financial value of their patent. They can thus further decrease their investment costs. At this point, it should be reiterated that in the last few years antitrust law has issued extensive case law on FRAND-compliant conditions.[1559] This could generously be transferred to compulsory licences under patent law.

2. Incremental Improvements

857 According to the interviewed software companies, there was another scenario where the quality ladder and path dependency was visible in software engineering. The software companies believe there is a social and market potential for ingenuity hidden in patent protected inventions, because some inventions could be further enhanced to achieve an even greater development.[1560] Although these enhancements could be of great use to society and the market, they would currently be partially impeded through patents. The companies therefore advocated a statutory provision according to which incremental improvements to an innovation should be legally enabled.

858 Patents, among other things, are intended to promote the progress of science and useful arts.[1561] However, contrary to copyright, today in patent law depen-

1557 Heinemann (2005), 76.

1558 See also remark in Art. 31 lit. k TRIPS Agreement.

1559 See above N 846, with further references to legal publications that discuss contemporary case law.

1560 See N 580 ff.

1561 See Art. I sect. 8 clause 8 of the U.S. Constitution.

dent inventions with novelty character and and their own inventive strength in general fall within the scope of the preceding patent and can thus be forbidden by the first patent holder.[1562] According to the prevailing jurisprudence, only if "the total of the technological changes" are *beyond* what the inventors disclosed, a significant advancement is available which is able to overcome the protection scope of a preceding patent.[1563] Even if an inventor "may have further developed the patent doctrine and the [inventor's] further development may even be patentable, (...) this does not lead out of the scope of the earlier patent".[1564] The protective scope of a patent thus is very broad compared with other exclusionary IP rights such as copyright. It should be pointed out that the right of the patent holder to exploit their invention is firmly established in law and was in principle not disputed in the interview study. However, some of the interviewed companies expressed their concern that patent holders could abuse core technologies and artificially prevent spillovers if an extensively wide scope of protection was assumed and enhancements were impeded.[1565] If third parties are prevented from developing their own inventions by an excessively broad protection scope of somebody else's patent, economic progress slows down.[1566] Compulsory licensing favouring incremental improvements to existing inventions could help to maintain the balance between fostering inventions and compensating society for the patent holder's exclusionary rights, while also meeting the difficulties of blocking effects in patent law. At the same time, the know-how contained in patented inventions could be made better useable by promoting further enhancements that are of value for society and the market.[1567] Subsequently, I would therefore like to discuss a statutory possibility for dependent inventors of incremental improvements to obtain a compulsory licence to use.

1562 See above N 268.

1563 *Texas Instruments, Inc. v. United States Int'l Trade Comm'n*, 805 F.2d 1558 (Fed. Cir. 1986), at 1571; similar explanation in BGE 142 III 772 c. 6.4.

1564 See the decision of the Swiss BPatGer of December 18, 2018, O2016_009, c. 48.

1565 See N 512 ff., N 580 ff. and N 653 f.

1566 Same conclusion in decision of the German BPatG of June 7, 1991, 3 Li 1/90 – *Zwangslizenz*, published in GRUR, 1994, 98 ff., 101 f.; see also LEMLEY/BURK, 92; HILTY (2018), 1186 ff.; OSTERRIETH, particularly 988 f.; see also dicussion in MARBACH/DUCREY/WILD, N 215; WOLFF, 40

1567 See similar thoughts in SCOTCHMER (2006), 132 ff. and 146 ff. regarding social value and improvement of ideas.

a) Insufficient Regulation in Art. 31 lit. l TRIPS

859 According to Art. 31 lit. l para. i TRIPS Agreement, inventions claimed in a second patent may obtain governmental authorization to use elements of a preceding patent if the second patent involves *an important technical advance* of *considerable economic significance* in relation to the invention claimed in the first patent.[1568] The provision also regulates under which conditions a "dependent patent" may be granted. At first glance, it may seem as if this provision was particularly designed to solve the aforementioned problem. But on a closer look, a number of difficulties arise:

860 According to the wording of the law, not every inventor of a dependent invention has the right to authorized use, but only a dependent patentee. In other words, the second invention must already have received a patent of its own in order to be able to benefit from the limitation in Art. 31 lit. l para. i TRIPS Agreement. The application of a purely dependent invention, however, will not have much chance if the relation to a predecessor patent is strong and obvious – which is often the case if the invention contains an incremental improvement to an existing invention and is mainly confined to it. A patent application is unlikely to be successful, or at least would not be recommended by a (patent) attorney to his or her client. The TRIPS-provision, or rather its implementation in the national statutes, thus does not serve as a promising aid to substantiate a new patent, but rather – if a patent actually has been granted – as means of defence against a former patent owner in civil proceedings of infringement.

861 Furthermore, according to Art. 31 lit. b TRIPS Agreement, the holder of the later patent must first attempt to obtain a licence from the first patent holder before seeking a state granted compulsory licence for use. This entails that the dependent inventor inevitably is going to appear on the first patentee's radar, and it may be expected that the latter would seek to prohibit the dependent

1568 See for example Art. 36 Swiss PatG, § 24 and § 85 f. German PatG, Art. L613-15 French IP Code; in contrast to other patent statutes, the U.S. patent code itself does not include a general compulsory licensing provision, but instead restricts their use to particular domestic statutes on individual thematic areas, such as plant variety protection and atomic energy. The European Patent Convention, as well, does not provide a provision for compulsory licences. However, a dependent patentee may rely on the provisions of national law (see decision of the German BPatG of June 7, 1991, 3 Li 1/90 – *Zwangslizenz*, at 101 f.

inventor from continuing its work via injunctions.[1569] While this problem in the area of standard-essential patents was partly countered in the European Union by means of antitrust law measures, there is no established jurisprudence on how to deal with injunctive reliefs in the case of incremental improvements and dependent patents.

The design of Art. 31 TRIPS thus may lead to a de facto exclusion of dependent inventors of an incremental improvement from obtaining a right to use. It can however serve as a basis for discussion in the following to circumscribe what an optimized regulation should entail. 862

b) Technical Advance

According to Art. 31 lit. 1 para. 1 TRIPS Agreement, the invention claimed in the second patent shall involve an important technical advancement in relation to the invention claimed in the first patent. The formulation "important technical advancement" represents an indefinite legal term. 863

Fortunately, the German and Swiss provisions implementing Art. 31 TRIPS are very similar to the international provision,[1570] so we can include their jurisprudence to further clarify what may be subsumed under the term: 864

A decision by the Commercial Court of the Canton of Berne in 2005[1571] held that the necessary *important augmentation for technology* would not presuppose an actual technological leap of progress. Instead, it could, for example, include that the new doctrine of technology *provided better means* which helped simplify or speed up a process or which was just less susceptible to in- 865

1569 This impression is reinforced by the fact that, with a view to Swiss practice, the existing provision in Art. 36 Swiss PatG has so far been used exclusively as a means of defence in pending patent infringement proceedings: See BGE 29 II 564; BGE 42 II 269 – *Transformer*; BGer of Januar 18, 1990 – *Doxycyclin III*, published in SMI, 1991, 198 ff.; decision of the HGer of the Canton of Berne of July 6, 2005, HG BE 6.7.2005 – *Anschlaghalter III*, published in sic!, 2006, 348 ff.

1570 Art. 36 Swiss PatG, § 24 German PatG.

1571 Decision of the HGer of the Canton of Berne of July 6, 2005, HG BE 6.7.2005 – *Anschlaghalter III*, published in sic!, 2006, 348 ff., with further references; apart from this decision, Swiss courts have only rarely been given the opportunity to apply Art. 36 PatG. This is despite the fact that the regulation for compulsory licences of dependent innovations had already been referenced in judgments of 1903: See BGE 29 II 564; BGE 42 II 269 – *Transformer*; BGer of Januar 18, 1990 – *Doxycyclin III*, published in SMI, 1991, 198 ff.; decision of the HGer of the Canton of Berne of July 6, 2005, HG BE 6.7.2005 – *Anschlaghalter III*, published in sic!, 2006, 348 ff.

terference. The advancement would lie in saving time or necessary steps, and hence in conserving resources. The court further held that progress could only constitute an enhancement if there was *a need for a substitute in an earlier patent*. It seems as if the court with the latter argument tried to factor in that some inventions exhibit a tangible potential for further development, while a voluntary, 'nice to have' progression was not worthy of protection. However, due to the consideration's unfortunate wording, the court's paraphrase could also limit the field of application for potential enhancements to strict replacements, not considering alternatives or an increase in efficiency on a voluntary research basis simply because the older invention did not show a "need for substitute". I am not convinced that, from today's perspective, the need for a substitute as a prerequisite does justice to the present market situation and to software as a potential field for digitalized inventions in particular. Still the court's consideration of a market need for progress, *in casu* the desire for a replacement, appears to have been reasonable in the specific case.

866 The Commercial Court of the Canton of Berne deliberately aligned its decision to an older case of the German Federal Court of Justice of 1970.[1572] The German Federal Court of Justice held that in order to invoke the German provision in § 24 German PatG the invention had to exhibit an *additional quality* other than being novel; it had to *enrich the technology* in order to be considered a relevant technical progress.[1573] In its decision it stated that the objects of investigation first had to be *similar* in order to conduct a comparison. The new improvement had to be set in an analogous technical context. Second, the new invention had to exhibit a *valuable property*. *The new means had to be better than the known ones* – the product or method in the new teaching had to exhibit advantages over the relevant teachings previously known in the state of the art of the according technology. Or the value of the invention could lie in the circumstance that the new doctrine did not provide the technology with better means, but with *further means*, e.g. where certain means or methods have already been established on the market, but another new one emerged. According to the court, this case would particularly apply to medicinal products. Even if a remedy only served as a "reserve product" that was available alongside oth-

1572 Decision of the German BGH of February 24, 1970, X ZB 3/69 – *Anthradipyrazol*, published in GRUR, 1970, 408 ff., c. 13, 14 and 18 ff.

1573 The decision dealt with pharmaceutical patents. As a side note, the argument of substitutability, to which also the Bernese court referred, seems to make more sense in this context of pharmaceutical patents because the pharmaceutical market is much older and more established than the software market where new alternatives and increases in efficiency still evolve.

ers, it could "help mankind out, if, for example, the others should fail or find obstacles which impair their use". It may also offer an *opportunity to choose* between several technical possibilities based on criteria such as expediency or different factual or local circumstances and needs. In the specific case of the German Federal Court of Justice, on therapeutic drugs, the court considered that the requested valuable property could be found in a better *technical effect* or in a *non-analogue technical constitution*.[1574] With this consideration, it opened up the application possibilities under the term "technical advancement" from sought substitutes to objectively justified alternative options and performance modifications.[1575] But the German ruling contained even more: it noted in the context of an *obiter dictum* that with the spread of technology in all areas of human life even today inventions could be made for which there was nothing "comparable" in the state of the art. The German court elaborated with foresight that in the light of more recent developments on the technology market, the concept of progress and its assessment by the authorities should be deliberately kept broad and open. With the strong tendency for digitalization,[1576] the court's statement may become particularly relevant for the field of apparatus patents, where, in the absence of distinctly similar points of comparison, the concept of technical progress and the relevant criteria must be analysed and defined anew.[1577] This makes it all the more important that Art. 31 TRIPS Agreement and its implementation in national statutes not only allows already patented inventions but also unregistered ones with incremental improvements. A deliberately open definition may support this.

1574 Decision of the German BGH of February 24, 1970, X ZB 3/69 – *Anthradipyrazol*, published in GRUR, 1970, 408 ff., c. 13, 14 and 18 ff.

1575 These holdings were confirmed in further decisions in Germany: decision of the BGH of March 14, 1972, X ZB 2/71, published in NJW, 1972, 1277 ff.; decision of the German BPatG of May 24, 1973, 16 W 72/72 – *Farbstoffbildungskomponenten II*, published in GRUR, 1974, 151 ff.; decision of the German BGH of March 13, 1984, X ZR 24/82 – *Chlortoluron*, published in GRUR, 1984, 580 ff.; decision of the German BPatG of August 2, 1982, W 49/80 – *Technischer Fortschritt*, published in GRUR, 1983, 240 ff; decision of the German BPatG of July 12, 2002, 14 W 51/01; decision of the German BPatG of February 7, 2017, 3 Ni 20/15. Particularly on the criterion of social usefulness, see: decision of the German BPatG of October 20, 1994, 23 W 6/93 – *Aussenspiegel-Anordnung*, published in GRUR, 1995, 397.

1576 See N 668 for further information.

1577 For difficulties in applying for application and process claims, see Beresford, N 4.45 ff.; Calame (2006), 664; Swiss Guidelines for the Substantive Evaluation of Patent Applications, 19, with notes.

867 In essence, the described meaning corresponds to that of incremental improvement as it was developed in the interviews. The software companies explained that the concept of incremental improvements should contain *a qualitative gain and* exhibit an *applicable added value to existing technical knowledge*, such as *better*, *more efficient*, *more effective or more flexible solutions to known technical problems*.[1578] The improvement may also constitute a *necessity for replacement* or for *additional alternatives* if other technological options are outdated and no longer technically feasible, if not sufficiently available or obtainable, or if a process is not always suitable. But *further enhancements* can also offer procedural or resource-oriented advantages, for example an increase in efficiency.[1579]

868 With regard to the fact that computer science and the software market are still evolving and are less researched than for example the pharmaceutical studies, the interpretation of the term technical advancement by the German Federal Court of Justice – allowing for new substantiated alternatives, performance enhancements and novelties beyond comparable markets – seems to be better transferable to software developments as understood in the interview series than in the decision of the Commercial Court of the Canton of Berne. It would be desirable if the concept of technical advancement were interpreted more broadly in Switzerland, similar to the German case law, in order to ensure technology neutrality and also enable new, but dependent, innovations in the emerging software sector. With this constraint, the definition and interpretation of the term 'technical advancement' could be adopted for compulsory licensing of incremental improvements.

c) *Important Advancement*

869 Not all advancements would justify a compulsory licence just "important" ones. However, this legal term is open to interpretation.

870 There is not much case law on how to interpret this criterion. According to the Commercial Court of the Canton of Berne,[1580] the term describes significant progress, which would not just be 'nice to have'. The technology must be ad-

1578 See N 465 ff. and N 470 ff.

1579 For example, if an invention would allow a process to be conducted with less effort. See for example decision of the OLG Duesseldorf of April 17, 1980, 2 U 106/79 – *Absatzhaltehebel*, published in GRUR, 1981, 45 ff., 49 f.

1580 Decision of the HGer of the Canton of Berne of July 6, 2005, HG BE 6.7.2005 – *Anschlaghalter III*, c. 2b, published in sic!, 2006, 348 ff., with further references.

vanced a great deal through the enhancement. Nevertheless, the court held, this would not necessarily presuppose a technological leap forward; a *considerable, remarkable development* would suffice. It appears that the court had difficulties in circumscribing what the term "important" actually involved. What we can take from this ruling is that advancement requires a certain *strength* and *significance*, a strength which *cannot be expressed in abstract terms, but must be outlined in each individual case* in light of the state of the art and the technical problem to be solved. It is thus *context-specific*.

Also according to the software companies, the concept of incremental improvement should involve some kind of "substance" and "material enhancement". The qualitative reference in the term "important" advancement here seemed to correspond to the concept of "incremental", i.e. augmenting improvement. The software companies specified that if a new idea brought an old invention to the "next level", such as better, more efficient, more effective or more flexible solutions to known technical problems a potential case of incremental improvement would be present.[1581] *The improvement must be of such importance that the contained value and benefit for the addressed users and market participants is evident and tangible.* The dependent invention has to *stand out* from the state of the art. The criterion of "important" advancement is, although somewhat vague, able in principle to be adapted to the proposed solution of compulsory licensing for incremental improvements and can consequently be transferred to it. 871

d) Considerable Economic Significance

The third criterion, that the advancement must be of "economic significance", represents an indefinite legal term which is difficult to grasp. In particular, it is disputed what economic term applies and whether business or macroeconomic criteria for the benefit of the market or society should be taken into account. 872

An older jurisprudence of the Swiss Federal Supreme Court[1582] held that public interests would not be the focus when evaluating the term of technical progress. Instead, only the concrete, individual circumstances should be considered.[1583] The social aspects would already be addressed by generally allow- 873

1581 See same argumentation in: SAMUELSON ET AL., 2346.

1582 BGE 29 II 564.

1583 BGE 29 II 564, at 577 f.; Heinemann instead seems to support the admissibility of dependent inventions on the basis of the public interest (see HEINEMANN (2002), 183 fn. 282).

ing incremental improvements to inventions. The court instead examined with the help of expert opinion whether the invention in question, a chair for schoolchildren, represented *real industrial significance* ("réelle importance industrielle") *in comparison with other furniture of the same kind*. It thus set the dependent invention in relation to other similar products (relative comparison method). It further specified that if the invention was particularly designed to be used by customers or consumers – *in casu* schoolchildren – a judicial evaluation should further include considerations on the intended users' needs.[1584] On this basis, the court came to the conclusion that in the case of the chairs for pupils, the dependent invention would indeed exhibit a relevant enhancement of economic significance. The Swiss Federal Supreme Court hence included *macroeconomic aspects* in its decision, while it shut out other social considerations.

874 In contrast, the Commercial Court of the Canton of Berne[1585] based its decision exclusively on *business management aspects*. In particular, it took into account whether there was a demand for the invention based on the catalogue prices and whether the products could be used as door openers to sell other products from the same supplier with less effort. The court concluded in this case that not only was there no technical need for a replacement but that this replacement did not constitute an economic significance that would justify a compulsory license. Business economic aspects were therefore considered to assess the later invention's market potential, including its potential for effective market penetration.

875 According to Swiss case law, the economic significance could thus lie in either, macroeconomic or business economic aspects. The inventor has to show that his/her invention involves added value either in terms of supply or demand markets or a positive effect on other market participants or users.

876 In comparison, the criterion of economic significance is interpreted rather broadly in German practice, allowing both business and macroeconomic argu-

1584 BGE 29 II 564; see also discussion in MAMANE, 31 f.

1585 Decision of the HGer of the Canton of Berne of July 6, 2005, HG BE 6.7.2005 – *Anschlaghalter III*, c. 2f, published in sic!, 2006, 348 ff., with further references.

ments;[1586] yet the criterion is also somewhat neglected in that it is seen as a minor factor relating to the evaluation of the technical advancement. When assessing the "success" of the invention, non-technical aspects such as economic value, market penetration or the satisfaction of a need are taken into account as evidence in favour or to the detriment of the required advancement.[1587] The criterion of economic significance, meanwhile, does not appear to be assessed separately, independent of the latter. This neglect of the criterion in German practice is not consistent with the legal wording of Art. 31 para. l TRIPS or § 24 German Patent Act that define the economic significance as a requirement of its own. In this respect, Swiss case law is more accurate, although still not fully clear on the interpretation of the term.

According to the software companies, the incremental improvement must exhibit an "added value" either for the market or for society. Although this argumentation favours the macroeconomic perspective, it is neither clearly prescribed by law nor absolutely necessary to limit the economic concept to either macroeconomic or business economic aspects. Both can substantiate significance as long as they do not solely serve the dependent inventor's interests alone, and hereby disproportionately and unduly distort the patent holder's exclusionary rights. A *minimal external impact on the economy*, other market participants or society should be required. The criterion of economic importance should therefore be retained. 877

With regard to demonstrability and numerical reference of the economic significance, another holding in the ruling of the German Federal Court of Justice[1588] should be taken into account. Hereafter, assessing the extent to which a new (inventive) drug and its freedom from side effects entails positive merits could only be achieved on the basis of several years of clinical testing. Such an 878

1586 See decision of the BGH of March 14, 1972, X ZB 2/71, published in NJW, 1972, 1277 ff.; decision of the German BPatG of May 24, 1973, 16 W 72/72 – *Farbstoffbildungskomponenten* II, published in GRUR, 1974, 151 ff.; decision of the German BGH of March 13, 1984, X ZR 24/82 – *Chlortoluron*, published in GRUR, 1984, 580 ff.; decision of the German BPatG of August 2, 1982, W 49/80 – *Technischer Fortschritt*, published in GRUR, 1983, 240 ff; decision of the German BPatG of July 12, 2002, 14 W 51/01; decision of the German BPatG of February 7, 2017, 3 Ni 20/15. Particularly on the criterion of social usefulness, see: decision of the German BPatG of October 20, 1994, 23 W 6/93 – *Aussenspiegel-Anordnung*, published in GRUR, 1995, 397.

1587 Decision of the German BGH of February 24, 1970, X ZB 3/69 – *Anthradipyrazol*, published in GRUR, 1970, 408 ff., c. 2.).

1588 Decision of the German BGH of February 24, 1970, X ZB 3/69 – *Anthradipyrazol*, published in GRUR, 1970, 408 ff., c. 13, 14 and 18 ff.

intensive evaluation should, according to the court, not be conducted by the patent granting authority. With this statement the court presumably wanted to express that, under the provision of § 24 German Patent Act, the authorities could only make an approximate *ex ante* evaluation of the possible advantages and consequences of an invention. Accordingly, an assessment for the individual case could only be made on the basis of an abstract judgement in the original situation, without taking into account long-term observations and studies. In the present understanding, *economic significance should therefore be shown in abstract terms for a particular context.* The dependent invention's considerable economic significance must be demonstrated by the dependent inventor in the individual case. This argumentation does not require empirical long-term data, as this would be too cumbersome and the threshold for proof would hereby be set too high. Rather the inventor of the dependent invention should show for the specific case with substantiated economic proof and expertise where the economic value of his or her approach lies in comparison to the known original invention and to what extent business or macroeconomic benefits may therewith be achieved.

e) Notes on the Possible Licensing Procedure and Terms

879 Having gone into more detail as to what could be understood by the concept of incremental improvements, or important technical advancements with economic significance, it should further be addressed under what terms these should be allowed. In this regard, in the main we could refer to the stipulations of Art. 31 TRIPS Agreement and its implementation in the various national legislations.[1589]

880 Art. 31 TRIPS Agreement enables dependent patent holders to obtain a *right to use*. They may apply for a specific legal licence that substitutes any arrangement with the original patentee. We are consequently, again, talking about compulsory licencing. That said, I am, however, unsure how far the compulsory licence should go, whether it should only include a right for use or to what extent this secondary invention should also be allowed to be commercialized and exploited independently. The interviewed software companies did not discuss this question.

881 From a macro-economic perspective, we may take into consideration the *degree of dependency* of the later invention on the first invention as well as the *exhibited strength of progress* and *its economic value*. The legal criteria can

[1589] See for example Art. 36 regarding Art. 40e Swiss PatG, § 24 German PatG.

therefore be applied not only to the question of granting the compulsory licence as such, but also to determine the scope of the licence and put it into proportion to the inventions' achievement in the individual case. The compulsory licence should be proportionate in the specific situation and not be more extensive than necessary to unfold the involved inventive potential. Another solution would be to follow the idea of *joint use* where the secondary inventor would have to back-license their invention to the original inventor, and in return receive compensation for their improving contribution.[1590] Compulsory back-licensing models would guarantee that incremental contributions were also legally recognized and incorporated. Hereby, ingenuity in research and development could be better supported without endangering the economic source of a patent holder. By imposing the duty on the secondary, recognized, inventor to inform and work with the primary inventor, he/she may use the invention but not exploit it on his/her own.

How exactly this process could look in practice should be further evaluated. 882
To some extent, we can make use of the experience gained in the field of standard-essential patents, e.g. with regard to the compensation and its judicial evaluation. On the other hand, we should actively address the question of whether the dependent inventor may seek a compulsory licence for his/her incremental improvement without obtaining a patent of his/her own first, and whether he/she should be allowed to apply for a compulsory licence prior to finding a solution with the patentee. The prior consultation of the patent holder could lead to hasty protective measures on his/her part. As mentioned before, there is no precedent on how to deal with injunctive reliefs in the area of incremental improvements. While one can imagine various solutions to this problem, such as mechanisms under antitrust law or via the principles of abuse of rights and acting in good faith, it would also be a practicable way to establish a mechanism *de lege feranda* within which an inventor of an incremental improvement could directly apply for a compulsory licence from a competent authority. This approach is compatible with Art. 5 lit. A Paris Convention, but would only comply with Art. 31 TRIPS Agreement if there was a case of urgency or increased public interest. Otherwise, prior consent of the patentee would be required.

[1590] SCOTCHMER (2006), 152 ff.; see also discussion in TROLLER (1985), 852 ff.

VI. Conclusion

883 Based on the findings of the interview study, this chapter has established to what extent the existing law is able to reflect the structures in the development and distribution of computer programs in the protective scope of copyright and patent law. It has appreciated what is working well and addressed where room for improvement is visible. It proposed solutions to selected problems and suggested rough models that could better integrate the needs and requests of the interviewed software companies in order to improve the current legal protection of computer programs with copyright and patent law. The following section will briefly summarize this chapter's main conclusions.

884 As *a first important conclusion*, we can state that in general some kind of legal software protection is required, but that there is *no necessity for a sui generis framework* to protect software adequately with law.[1591] The work processes and the software companies' needs and wishes could be incorporated into the current *copyright and patent law regime*. I was quite surprised to discover how well established the *hybrid software* protection regime in intellectual property law is, and that the software companies, in fact, appreciate both copyright and patent law. It was found that the two different institutions largely complement each other in the field of software due to their different subject matters, and that this combination satisfies different basic needs of the developers. Patent law is designed to capture specialized domain know-how that is reflected in inventive technical ideas and implementations as it particularly addresses the technical teaching or solution that is disclosed with the publication of a patent application. Copyright, on the other hand, is able to protect the resources and labour that were invested to elaborate a creative outcome, including its conceptualization and realization. It perfectly shelters expenditures made, whose result can easily be imitated when it is clear what a successful and working version of the component could look like. Accordingly, computer programs should also in future be protected with copyright and patent law. While copyrighting is, markedly, well accepted among software companies, and only turned out to require a little fine-tuning in order to respond to today's evolution, patenting is in a more difficult position. In the United States software manufacturers, market participants and consumers talk about 'over-patenting', and the negative effects this may entail. The excess of (trivial) patents and the potential blocking effect this causes has an immense impact on the international debate of patenting software. Although hardly any of the software com-

[1591] For more information, see Chapter 6 Section I and II.

panies reported having negative experiences in this area, the opinions presented were strongly influenced by it. In Europe, on the other hand, the European Patent Convention in particular fails to fully recognize software as a patentable invention. The interviewed software companies expressed that it would be absurd of the European patent authorities to require 'a further technicity' for computer programs. Every to-be-patented invention has to involve a technical problem and provide a technical teaching. Without 'something in addition' to the basics, neither a software good nor a mechanical one would meet the technicity requirement, and would thus not fall under the subject matter of patent law as defined in the international treaties and national statutories. Software companies have thus strongly advocated that computer programs should not be treated differently from other inventions. Computer programs should, consequently, also be eligible for patenting if they fulfil the classic patent requirement and fall under the subject matter. We should therefore revert to the basics of copyright and patent law and not create additional hurdles or initiate a new legislative solution. *With the existing fundamentals in copyright and patent law, software can be dealt with sufficiently.*

Second, the existing fundamentals regarding the subject matter should be reviewed with a fresh eye in order to technically interpret software in a modern and dynamic way.[1592] The goal of copyright and patent law is to shelter creative or innovative developments. It was outlined that creativity in software engineering determines how a problem is solved and how a solution is implemented. It involves artistic questions, but also smart solutions for business or technical problems. Creativity therefore represents the result of a human play. There is room for creativity where the scope for productive decision-making out of the ordinary is available and the personality of the author can have a stylistic effect. On the other hand, inventiveness is entailed, if from the perspective of an initial prior position, the presented process or good represents progress or incremental novelty. In the field of software development, inventiveness may firstly, as in every technological field, be found where software can be used to develop an entirely new invention. Further, in the context of digitalization, older inventions that were developed with mechanical governors or electronic circuits can be reintroduced with an inventive teaching or solution. Finally, as software development is still a rather new technical field, the way that software is developed and data are processed opens another potential field for inventiveness. Based on these insights into what creativity and inventiveness mean in software engineering, it was established what should be 885

[1592] For more information, see Chapter 6 Sections III.A. and III.C.

protected with copyright and patent law. There are several software components that are typically included in a software product. From the perspective of intellecual property law, today, computer programs are mainly understood as an intellectual good or process that is expressed in a linguistic command form. The visual appearance of software, for example in the user interface or the look-and-feel, has been widely neglected in copyright. While the World Copyright Treaty statutorily limited copyright protection to the literary form of computer programs,[1593] we can assume that today's interpretation of the general copyright criteria also includes visual components in a computer program. The previously narrow scope of copyright law should thus be interpreted more broadly to further parts of software so we can better protect these visual elements. Furthermore, structural and organizational elements in the source code should be accorded greater consideration. Finally, it was debatable as to whether the development documentation was eligible for copyright. Following the approach advocated in this thesis, it should likewise be protected with copyright if it fulfils the protection requirements and is sufficiently concrete. On the other hand, patentable teachings solving a certain problem are largely limited to the field of algorithms, if permitted at all. According to the presented opinion, in future, functional elements of the look-and-feel as well as functions and features with a clearly defined scope for implementation should be integrated into patent law in addition to algorithms.

886 Third, I came to the conclusion that the existing *protection requirements* in patent law *are suitable*, if tested substantially, for the technical specificity of software, and, at the same time, offer a sensible barrier to potential misuse.[1594] The internationally established criteria in copyright and patent law are dynamic enough to encompass computer programs and should hence be maintained. Nevertheless, I suggested *several small improvements to the definitions of the established protection criteria in patent law* that would help to better capture software as a creation in IP law. They could also be adapted for patent law in general and do not necessarily have to be restricted to the invention subject of 'software'. Firstly, the *expert figure concept* under the non-obviousness criterion has so far been interpreted in a way so it is hypothetically based on an 'average' skilled person with the capability to think logically. The findings of the interview study suggested that the expert should not just possess 'ordinary skills' but should rather show special skills in the particular science in question. The expert figure could be upgraded to help reduce the occurrence

1593 See N 262.

1594 For more information, see Chapter 6 Section III.B.

of trivial patents. In general, I believe the criterion of *non-obviousness* will become more significant for the evaluation of software patents in future in order to rule out trivial inventions. It has been observed that when examining the substance in patent applications, the necessary prior art to exclude trivial inventions is often missing. Accordingly, it was proposed to adapt the understanding of the expert term in such a way that an invention should be regarded as non-obvious if an expert with special skills in the particular art of software engineering when given the same task could not come up with the way in which the inventor had solved the problem within a reasonable time limit. This would increase the quality of the patent evaluation and trivial patents could be recognized more easily. Finally, the software companies suggested that an innovation should not just need to involve an applicable teaching but should also be useful to the market or society. The software companies thus asked for *added value* in the invention. It was discussed that instead of introducing a new criterion to implement this improvement, the added value of an invention could, for example, be examined in the evaluation of the subject matter or under the applicability criterion. I personally would prefer its implementation to be under the subject matter of an inventive teaching, because it could be approached earlier in a substantive check of criteria and this approach would also indicate the importance of the minimum quality a patentable object should exhibit. It would therefore represent a suitable measure to tighten the patent scope.

Fourth, *new development trends* in the software market, such as spiral and incremental development in agile approaches and continuous delivery, *can and should be integrated in copyright law*.[1595] However, this would require a more flexible and revised interpretation of the statutory provisions. So far, nobody has established whether and how these modern development methods are to be interpreted in terms of copyright and how they may affect the protection scope. One difficulty lies in the fact that due to their dynamic characteristics, these development methods contradict the classic understanding in copyright of seeing a work as a good with static and fixed features. Lawyers have tried to address this problem by (over)regulating the issue in contracts. It was suggested that spiral development, incremental extensions to and changes within previously released works could be integrated doctrinally in copyright, without requesting any regulatory revisions. 887

The rise of new technological possibilities and prevailing trends in software development further make it necessary to *reinterpret certain elements which* 888

[1595] For more information, see Chapter 6 Section III.E.

are by legal definition excluded from copyright protection.[1596] This is particularly true for copyright-hostile *ideas and functional elements* and *creative works that lack originality*. Their legal understanding should be modified in accordance with current circumstances and tailored to their current field of application, i.e. software. According to this understanding, technical necessities, engineering standards and business best practices are often functionally predetermined or utilitarian, which is why they are often not eligible for copyright protection. Further it was explained how copyrightable expressions can be distinguished from non-copyrightable ideas in software development, and how particular details and additional particularities included in a development could enrich an original contribution. Based on the established merger doctrine, the abstraction test and the current scènes-à-faire practice, I proposed a rule of thumb called the *blackbox test* to assess software's eligibility for IP protection.

889 Fifth, for computer programs we face major difficulties in determining the *starting point for legal protection in copyright*.[1597] In the past, it was usually possible to tie it to the releasable major or minor version of the software creation. Where new development and commercialization approaches are applied, this clearly fixed and easily determinable starting point for protection has blurred. In practice, only the implementable components were protected as ready-to-use results, creating a protection gap for all expressions that have reached their creative threshold earlier and thus would have fulfilled the copyright requirements before. It was highlighted that, instead, protection should start on the development timeline when a mature concept is available and the expression was sufficiently specified. Through this, the people responsible for software projects could be better protected from interventions of third parties, particularly in the case of commissioned work where developments are frequently shared without sufficient legal protection. The protection gap could thereby be closed. For this purpose, I outlined how we can assess when a computer program is specified enough so that it can be granted protection at the earliest possible opportunity. It was further elaborated where the starting point for protection could be set appropriately dependent on the particular development model in question.

890 As a sixth conclusion, and closely related to the last one, it was found that the term of protection in both copyright and patent law of 50 years after the last author's death and 20 years after a patent application was filed, respectively,

[1596] For more information, see Chapter 6 Section III.D.

[1597] For more information, see Chapter 6 Section IV.A.

may entail an excessive defence effect with regard to new technical possibilities, evolved practical circumstances and market trends.[1598] The interviewed software companies emphasized that the provided term of protection would currently be too long compared to the estimated software product life cycles. Thus, contrary to the purpose of IP law, copyright protection would expire and the work would only be made available to third parties when the software was significantly outdated and barely usable. This would artificially and unnecessarily reinforce the monopoly-like, exclusionary right in IP. It was therefore suggested to shorten the protected durations of copyright and patent law to 20 and 10 years. However, the data basis for this question was somewhat thin, so this aspect should be challenged and verified with more data. Even without a binding statement on the exact possible term of protection, this conclusion can be regarded as a clear rejection of the existing long terms of protection. Furthermore, tying the term of protection to the author's death in copyright for a good that is of more economic and technical relevance than artistic was deemed inappropriate. For this reason, I recommended altering the connecting factor in copyright and linking the term of protection, as with patent law, to the creation or the release of the particular work (module) in question. Both these proposals would help to better reflect the object of protection software.

Seventh, and as a small excursus, the availability of new technological possibilities has made it easier and more efficient to copy and imitate third-party software components, which is why it is important to continue monitoring the legal treatment of *second-hand works*, particularly in copyright.[1599] I explained that the current jurisprudence to evaluate third-party derivatives in classic copyright law would be sufficient to manage similar cases with software works. At the same time, the interview study showed a high degree of uncertainty about whether *translations* from one programming language to another constituted a copyright infringement. In the case of human languages, translations fall under the exclusion right in the copyright scope of the primary author or right holder. Translating code between programming languages without consent should likewise represent a copyright infringement if the derivative lacked sufficient originality. It would be desirable to introduce an explicit legal regulation in copyright law regarding translations of literary works in computer programs in order to eliminate existing ambiguities and doubts. 891

[1598] For more information, see Chapter 6 Section IV.B.

[1599] For more information, see Chapter 6 Section V.A.

892 As the eighth and final conclusion, also as an excursus, it was found that software development particularly relies on a quality ladder and exhibits a path dependency, which is why it has become necessary to try and improve the way patent law deals with successor developments and second-hand inventions.[1600] The software market shows a great tendency and need for standardization. Where a *standard essential invention* is present whose further use by third parties is blocked by an exclusionary right, such as a patent, other market participants are obstructed. Over the last decade, authorities and courts have tackled this problem with antitrust law. However, since the problem of a potentially blocking effect of a monopoly-like exclusionary right has its origins in patent law, I am proposing a solution under patent law with the introduction of compulsory licences for standard essential patents on the basis of existing international regulations. It was further found that impulses in market-established inventions could be used to drive research and achieve an even greater, *incremental improvement*. Although this would be of great use to society and the market, it might be impeded through patents. It was explained that although there is a legal possibility for compulsory licensing of derivative inventions, it has hardly been used in the past, moreover, it could only be applied to second inventions that were already patented. Accordingly, I proposed introducing compulsory licences for innovations that contained an incremental improvement to patented inventions.

[1600] For more information, see Chapter 6 Section V.B.

Chapter 7: Prospect and Closing

This final chapter reflects the possible implications of the thesis. It elaborates 893
on the question to what extent the results of this work permit substantial conclusions and contribute new concepts to the current status. It also discusses possibilities for further research. The chapter notes where *de lege lata* copyright and patent law exhibit the potential to better address and incorporate the needs and wishes of software companies through a contemporary interpretation or a change in practice. In a separate section it points out where a revision in law has become advisable and which topics could be addressed and in what way.

I. Scientific Contribution

This doctoral thesis aimed to shed light on the phenomenon of software de- 894
velopment and commercialization from a legal perspective, but also wanted to leave the safe haven of jurisprudence. The complex and peculiar processes surrounding the creation of software were not to be characterized purely unilaterally from a legal perspective, but were to be fundamentally researched with the help of socio-scientific methods and in close cooperation with representatives of the software industry. On the basis of these results, the scope of legal software protection in copyright and patent law was then reviewed and assessed.

With this work I would like to contribute to the field by researching an unfa- 895
miliar study environment for lawyers. The main achievement of this thesis is to provide the necessary basic scientific research and data on the question of how the software industry works today and how this affects the protective scope in copyright and patent law. Using socio-scientific methods, it establishes a theoretical understanding of classic practices as well as market-specific peculiarities of software development and commercialization. From a legal perspective, it provides a different understanding and new points of view on existing doctrines. It further recognizes and resolves some prevailing trends in the software industry, which have so far been largely unaddressed in copyright and patent law. It aims to suggest practical approaches as to how certain problems in software engineering could be better addressed and incorporated in law. It also makes a scientific contribution to the international debate on the eligibility of computer-implemented inventions for patenting. The results of this dissertation may help practising lawyers, authorities and

courts to better understand existing practices in software development and commercialization, to express them in legal terms and interpret them with a contemporary understanding suitable for the practical circumstances and market trends. At the same time, this work identifies a need for action and the potential for improvement for legislative bodies. The observations could be applied to any modern copyright and patent law system, although gathered specifically for the software market.

II. Key Merits

896 This work tries to describe favourable framework conditions with regard to the copyright and patent law scope that are consistent with the practicalities and most important basic needs of the software industry. The first merit it offers is *new and scientifically based insights into the development and commercialization processes of the software industry* and the guiding technical ideas behind them. In addition to the classical linear development process from ideation to realization to implementation, the work takes up various modern development concepts such as incremental or agile techniques and elaborates legally relevant particularities in incorporating them. Also new approaches such as continuous delivery, which have not yet been addressed in law, are discussed in detail. I explored to what extent and when creativity and inventiveness may play a role in software engineering. The work also deals with recognized but also newly significant software components, such as the look-and-feel. Furthermore, time-relevant factors such as the time-to-market and the duration of the software lifecycle are considered. It also deals with contemporary commercialization approaches such as software as-a-service. A second merit of this work is that it *provides important information on how to evaluate the scope of legal protection in copyright and patent law for software* by integrating the findings of the socio-scientific analysis into law, in particular with regard to the requirements of protection and the term of protection. The most important finding of this thesis was that today's hybrid system with copyright and patent law adequately serves software as an object of legal protection and provides the necessary basic conditions and can serve as the basis for future legal software protection. Thus, this thesis elaborated several new ideas and models based on the current legal regulation, interpreting within the boundaries as well as testing them. At the same time, it was pointed out where software, as a dynamic and digital good, requires special protection and where further regulatory action is needed. This thesis elaborated possible solutions for an optimized substantive examination of software to address some con-

cerns about the negative effects of patents on the market and its impact on inventiveness. It was found that rigidly protecting solely the final working products of software engineering, such as developed algorithms and implemented source codes could not do justice to the modern, dynamic software development and commercialization approaches. This work critically questions known processes and their legal regulation under copyright law. Through an examination of the technical factors and project organizational considerations, it offers different concepts to integrate new development methods into copyright, to address traditional interpretations of the copyright scope.

III. Area for Action De Lege Lata and De Lege Ferenda

Based on the finding that there is no need for a *sui generis* law and that the current copyright and patent law largely provide the necessary foundations, my proposed solutions to unresolved legal questions whenever possible remained within the existing legal structures. Where feasible, copyright and patent law have therefore been reinterpreted *de lege lata*. Where the law would have been strained, I made proposals for revisions in law *de lege ferenda*. 897

A. De Lege Lata

Every legal system must remain dynamic and flexible in order to respond to social change. In the case of software development and commercialization, it was found that copyright and patent law can provide the necessary legal framework for the market and reflect the predominant circumstances. Through a modern legal interpretation, the current structures and actions can be represented adequately. There is no need for sector-specific exceptional rules, but it is important to newly interpret the complex economic and technical interrelationships for law in the perspective of current trends. I recommend that the following points are tackled in a reinterpretation *de lege lata*: 898

First,[1601] it was found that, in the past, the field of application in copyright and patent law has been limited to specific software components. In copyright law, for example, the scope of protection has been limited to one-to-one pirate copies and literary elements such as the source code. It was suggested to expand copyright protection to organizational elements in the software, such as structural code elements, visual elements in the graphic user interface and the look-and-feel. In addition, other forms of expression that illustrate creative 899

[1601] For more information, see Chapter 6 Section III.A. and III.C.

ideas in a sufficiently concrete manner should also be eligible for copyright protection, if they fulfil the protection requirements. This is particularly true for the development documentation, as it may contain crucial creative ideas in a first intermediate working result. Similarly, in patent law, elements other than just the algorithm should be patentable, such as functions and features and functional elements of the look-and-feel.

900 Second,[1602] it has been shown that although the protection requirements in patent law are adequate, significant terms should be defined differently to better integrate software into law. The suggested adaptations can also be extended to other subjects of protection. Specifically, it was pointed out that the 'expert figure' required to assess the concept of non-obviousness in patent law has been based on a person with average skills. The interview findings suggested that the expert should possess special skills in the particular science in question. Consequently, an invention would be non-obvious if an expert with special skills in software engineering when given the same task could not come up with the way in which the inventor solved the problem within a reasonable time limit. The interviewed software companies also argued that software should only have patent protection if it offered added value for a third party, for example the customer or other market participants. The criterion of added value does not yet exist in law, but could be integrated under the term of inventive teaching.

901 Third, new and dynamic forms of development in software engineering, such as spiral and incremental development and continuous delivery, challenge the rigid, one-cycle-oriented copyright. The fact that a work may undergo further changes after its first publication and may both alter internally and grow externally is something completely foreign to copyright.[1603] However, both types of change can influence the work consistency and thus the scope of protection. Following the approach advocated in this thesis, incremental extensions and inner changes within a previously released work can be incorporated into the current copyright doctrine, but require some careful thought. It should be considered how the scope of protection of the work as a whole is affected by the increasing or reduced originality of individual elements. However, elements that are legally excluded from protection, such as copyright-hostile ideas and functional elements as well as creative works that lack originality, should also be interpreted in a contemporary way in order to do justice to the

1602 For more information, see Chapter 6 Section III.B.

1603 For more information, see Chapter 6 Section III.E.

content of the restriction.[1604] It was noted that technical necessities, engineering standards and business best practices are often functionally or socially predetermined by the idea to be realized, which is why they are often not eligible for copyright protection. Further, it was explained how copyrightable expressions can be distinguished from non-copyrightable ideas in software development, and that particular details and additional particularities could enhance an original contribution.

Fourth,[1605] it was emphasized that software, relative to other work categories, exhibits a creative threshold with a lot of conceptual details at a comparatively early stage in the development process, e.g. in a draft. Moreover, due to its economic rather than artistic features and the strong tendency for cooperations in software development, working concepts are usually shared at an earlier stage with potential collaborators. Software, as an accessible and copyable good, makes it vulnerable to infringements. With a contemporary interpretation of the existing rules, it would certainly be possible to turn away from the traditional protection of results in copyright and set the starting point of protection at an earlier stage in the development process, possibly as soon as the work has reached appropriate maturity. 902

Fifth,[1606] the availability of new technological possibilities has made it easier to copy and imitate third-party software components. One new phenomenon linked to this development is the rise of unapproved source code translations from one programming language into another. While the author's right in general covers translations, it is highly debatable whether this applies only to human language or could also cover programming languages. In this case a modern interpretation of the existing legal regulation could close some gaps. 903

Finally,[1607] the exclusive character resulting from patent law can have a large negative effect on the market, particularly when it comes to standard essential patents. I proposed introducing compulsory legal licences to resolve this issue. It seems entirely feasible to solve this problem on the basis of existing international regulations by reinterpreting the concept of public interest in Art. 8 in conjunction with Art. 31 and 31bis TRIPS Agreement in a contemporary manner. 904

1604 For more information, see Chapter 6 Section III.D.

1605 For more information, see Chapter 6 Section IV.A.

1606 For more information, see Chapter 6 Section V.A.

1607 For more information, see Chapter 6 Section V.B.1.

905 All these reinterpretations of the existing regulations would make a significant contribution to improving the integration of software into copyright and patent law. In the absence of new regulatory efforts, they can be easily implemented and could also be adopted in the new practices of authorities or courts. It would therefore be an easy way to address the existing needs and wishes of the software companies in a prompt and cost-effective manner.

B. De Lege Ferenda

906 Where the law neither provides the necessary foundations nor, due to a rigid framework, permits reinterpretations, regulatory intervention is required. I am aware that a revision of the law in the highly internationalized field of intellectual property law would be tedious and that political constraints further complicate the matter. This work has therefore attempted to limit proposals for regulatory revision to where they appear unavoidable or urgent, and to ensure that they are target-oriented and practicable. The following topics should be addressed by amending the law *de lege ferenda*.

907 First,[1608] the most urgent and probably greatest need for legislative action lies in lifting the exclusion of computer program patents in the European Patent Convention. The software companies clearly favoured the patenting of software despite its possible disadvantages. The European practice to date has been inconsistent and largely unpredictable for potential patentees. Additional hurdles were created for the legal assessment of computer programs that, firstly, contradict the wording of the law in the European Patent Convention and, secondly, are senselessly redundant, because the possible negative effects and potential hazards of software patenting could be adequately countered with a substantive assessment of the patent requirements. Thirdly, it artificially treats software as a potential object of protection unequally for no apparent reason. The European practice is no longer appropriate and appears antiquated. It goes against the fundamental need and right of software developers to be able to protect their inventions with patent law, provided that they meet the patent requirements. It is thus time to finally cut out this outdated provision as there is no reason to continue with it. Even if this effort to abolish the software patent exclusion is considerable, it is urgently necessary to revise Art. 52 para. 2 of the European Patent Convention.

[1608] For more information, see Chapter 6 Section II.

Second,[1609] for clarification purposes, copyright law should explicitly include the development documentation as a copyrightable expression. Although it represents an early working product in the software development process, it was found to be subject to third-party infringements and should thus be legally sheltered, if it is tangible and sufficiently developed to test its copyrightability. Also, the author's right of determination in the field of translations should be expressly extended to computer program languages. I therefore recommend integrating both of these small regulatory changes into the international treaties. However, as it is compatible with existing international legislation, it can also be introduced directly at a national level. 908

Third,[1610] it has been shown that there is a distinct time discrepancy between the expected life cycle of a computer program and its protection under copyright and patent law. In the case of software, creations are only made available to third parties when they are completely out-dated and hardly usable. The function of intellectual property rights, where a creation is subject to exclusionary rights, is that the creation should be made available to society and other market participants after a certain period of time during which the creator is able to recoup their investments but it is still usable for society. The long terms of protection – 50 years after the death of the last author in copyright, or 20 years after the filing of a patent – are disproportionate and constitute an excessive measure of protection. In view of the data provided by the interviewed software companies, it was proposed to shorten the term of protection for stronger intellectual property rights such as patents to 10 years after disclosure, and the term of protection for weaker IP rights such as copyright to 20 years after publication. Due to the small sample, however, it is difficult to make quantitive statements regarding protection duration. Nevertheless, a clear need to adapt the term of protection to the life cycles emerged. Changing the term of protection for copyright and patent law would require the adaptation of a number of international agreements governing the maximum durations in copyright and patent law, including Art. IV para. 2 UCC, Art. 7 para. 1 RBC and Art. 12 and 33 TRIPS Agreement. However, this revision would be important in order to achieve a proportionate scope of protection. It was shown that software has more economic and technical relevance than artistic. Consequently, tying the term of protection to the author's death in copyright establishes a personal reference to each developer involved in a software project which, considering its characteristics and the long duration 909

1609 For more information, see Chapter 6 Section III.C.7.

1610 For more information, see Chapter 6 Section IV.B.

of protection, is neither reasonable nor practicable. It would mean that each element of a component would have to be allocated to the employees involved and their life paths would have to be actively monitored by the company. As this procedure is not able to meet the needs and desires of the software industry, I would recommend altering the connecting factor in copyright and linking the term of protection, similar to patent law, to the creation or release of the particular work (module) in question. But as most national statutories currently do not relate to a specific creation to determine the term of protection, the legal provisions would have to be nationally revised.

910 Fourth, and finally,[1611] it has been shown that the exchange of know-how and cooperation in the software industry are comparatively high. This leads to a strong path dependency in the development of computer programs. This becomes particularly apparent in the problem of dealing with standard essential patents, most of which have so far been addressed by means of antitrust law. There is an urgent need for the handling of standard essential inventions to be legally improved in the software industry. I propose solving this problem through compulsory licensing in order to achieve a patent solution for a patent problem. In my view, the suggested approach is compatible with the existing legislation in Art. 8 in conjunction with Art. 31 and 31bis TRIPS Agreement, but it would require a contemporary interpretation of the concept of public interest. If, however, against expectations it did not correspond, legal revision would be possible at national level pursuant to Art. 5 lit. A Paris Convention. It was found that the access to and use of patents already granted should be improved, provided that inventions building on them showed incremental improvement. Here, again, a legal regulation at international level already exists. However, it only applies to second inventions that have already been patented. Applying for a dependent successor patent without holding the licence of the primary patent is, from a legal perspective, very tricky and not to be recommended, as well as hardly practicable. It is very probable that the succession invention would not be accepted by the patent issuing authorities, if it was examined, or would bethrown out of court. Accordingly, I proposed, in line with the existing provision for dependent patents, the introduction of compulsory licences for innovations that contain an incremental improvement on patented inventions. For this purpose, Art. 31 TRIPS Agreement and the national statutories building on it would have to be revised.

911 The suggested revisionary projects would help to resolve certain legislative issues that go against the practicalities and the needs and requests arising in

[1611] For more information, see Chapter 6 Section V.B.1. and V.B.2.

software engineering, such as the exclusion on software patents. Lifting the ban on computer-implemented patents under the European Patent Convention as well as introducing compulsory licences for standard essential patents and allowing incremental improvements would address the urgent needs of the software industry. Other issues, such as explicitly incorporating the development documentation and translations into the copyright code have more of a signalling effect. Likewise, reducing the term of protection is not essential, but should be addressed in order to guarantee a proportionate scope of protection and restore balance between the different interests behind intellectual property law. Irrespective of the urgency, all the points mentioned in this section should be addressed by the legislators in the near future.

IV. Limitations and Perspective

The present doctoral thesis and its research serves as a basis for discussion. Various hypotheses could be drawn up regarding the practicability of software development and commercialization as well as the legal requirements in dealing with them. Many of the results represent distinct tendencies, which have also been confirmed with member validation and are therefore well founded. I have pointed out in this thesis where contradictory statements or weak samples were found. In these situations, I tried, where possible, to give further insights, but at the same time disclosed weaknesses. This was the case, for example, when the interviewed companies were asked to give more details about the life cycle of their software products and services. Although they were able to provide information about their own products and services, as the software was often still running, only approximate values could be given. In addition, the sample of 12 interviews was too small to allow reliable quantitative data analysis. Nevertheless, the information provided clear trends that could be used to make statements. It would be useful to continue the research with a quantitative analysis, and to statistically verify the data found. 912

The comparison of socio-scientific findings with the current legal situation revealed where the framework conditions correspond to the needs and requests of the software companies, where certain gaps can be closed by means of *de lege lata* interpretation, and where there is room for regulatory action. The results of this analysis were then compared with the literature review, verifying them in comparison with alternative theories. In certain areas, for example the idea of introducing compulsory licences, a large amount of literature and jurisprudence exists which was used to review and critically question the results of this thesis. Where sufficient material was available, it was necessary to 913

choose which contributions were discussed. In other areas, there was a lack of literary or jurisprudential discussion, for example on the impact of new development methods on the scope of protection in copyright law, or the assessment of continuous delivery approaches under copyright law. Hence, I often had to look for potential solutions by studying the legal texts or exploring beyond existing frameworks. Numerous proposals and models were developed independently, which could not subsequently be validated because there was nothing to compare them with. It would be useful to be able to check these theories and models for consistency.

914 Intellectual property law aims to account for a balance of different interests, but this work, due to its scope, mainly dealt with the perspective of software companies. It would therefore be useful to also study the interests of other market participants, such as those of the successor market or the consumers. Special interests such as those of small- and medium-sized software developers could also be more closely examined in separate studies. These interests are particularly significant for the design of legal limitations and exhaustion which is why the discussion of this was not extensive in this work. Other important topics such as the role of procedural measures in the design of a fair patent system could only be touched upon. It would be interesting to explore these topics as a follow-up to this study.

Curriculum Vitae

Sarah Leins-Zurmuehle studied law and graduated from the University of Zurich in October 2013. Sarah worked in various positions at different IP-specialized law firms, district courts as well as at the University of St. Gallen. In the course of writing her doctoral thesis at the University of Zurich, she spent two months at the Berkman Klein Center for Internet & Society at Harvard University as a visiting researcher, before submitting her thesis in late 2019, which was accepted in May 2020. Sarah is accepted to the Zurich Bar and now works as an accredited (business) mediator and attorney.

The software industry is regarded as one of the most creative and dynamic industries in the world. At the same time, sheltering software through copyright and patent law has been a major point of contention for the past 40 years. This doctoral thesis aims to provide new insights to this discussion. Through the use of sociological methodology, it supplies the necessary basic scientific reasearch regarding how software is developed and commercialized nowadays. Based on these findings, it then legally evaluates to what extent copyright and patent law are able to reflect these structures and determines how an optimal protection scope for computer programs could look like today. This doctoral thesis on one hand offers novel insights and points of view on existing legal doctrines. It further acknowledges as well as legally qualifies some prevailing trends in the software industry, such as Scrum and continuous delivery, that have so far been largely unaddressed by copyright and patent law.