

Christoph J. Stettina  
Juan Garbajosa  
Philippe Kruchten (Eds.)

LNBIP 475

# Agile Processes in Software Engineering and Extreme Programming

24th International Conference on Agile Software Development  
XP 2023, Amsterdam, The Netherlands, June 13–16, 2023  
Proceedings


 Springer


OPEN ACCESS


# Lecture Notes in Business Information Processing

475


## Series Editors

Wil van der Aalst , *RWTH Aachen University, Aachen, Germany*

Sudha Ram , *University of Arizona, Tucson, AZ, USA*

Michael Rosemann , *Queensland University of Technology, Brisbane, QLD, Australia*

Clemens Szyperski, *Microsoft Research, Redmond, WA, USA*

Giancarlo Guizzardi , *University of Twente, Enschede, The Netherlands*

LNBIP reports state-of-the-art results in areas related to business information systems and industrial application software development – timely, at a high level, and in both printed and electronic form.

The type of material published includes

- Proceedings (published in time for the respective event)
- Postproceedings (consisting of thoroughly revised and/or extended final papers)
- Other edited monographs (such as, for example, project reports or invited volumes)
- Tutorials (coherently integrated collections of lectures given at advanced courses, seminars, schools, etc.)
- Award-winning or exceptional theses


LNBIP is abstracted/indexed in DBLP, EI and Scopus. LNBIP volumes are also submitted for the inclusion in ISI Proceedings.

Christoph J. Stettina · Juan Garbajosa ·  
Philippe Kruchten  
Editors

# Agile Processes in Software Engineering and Extreme Programming

24th International Conference on Agile Software Development  
XP 2023, Amsterdam, The Netherlands, June 13–16, 2023  
Proceedings

*Editors*

Christoph J. Stettina   
Leiden University  
Leiden, The Netherlands

Juan Garbajosa   
Universidad Politécnica de Madrid  
Madrid, Spain

Philippe Kruchten   
University of British Columbia  
Vancouver, BC, Canada



ISSN 1865-1348

ISSN 1865-1356 (electronic)

Lecture Notes in Business Information Processing

ISBN 978-3-031-33975-2

ISBN 978-3-031-33976-9 (eBook)

<https://doi.org/10.1007/978-3-031-33976-9>

© The Editor(s) (if applicable) and The Author(s) 2023. This book is an open access publication.

**Open Access** This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this book are included in the book's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the book's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG  
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Preface

It is with great pleasure that we introduce this volume of research proceedings of XP 2023, the 24th International Conference on Agile Software Development, organized on June 13–16, 2023 in Amsterdam, the Netherlands.

The XP conference has become the premier conference to combine both research and practice in the field of Agile software development, and beyond. It is a unique forum where Agile researchers, practitioners, thought leaders, coaches, and trainers get together to present and discuss their most recent innovations and research results.

The theme for XP 2023 was “Whole Team Sustainability”, a many-faceted theme centering on the need to band together to solve the complex organizational and environmental challenges of our time.

This year the Research Papers track received 40 submissions. Based on a thorough review process 12 papers – 11 full papers and 1 short paper – were accepted to be presented at the conference.

The research proceedings of the XP 2023 conference showcase a wide range of research topics related to Agile software development. The papers in this volume cover various aspects of Agile software development, including: Sustainability, Engineering, Transformation & Mindset, Organizational Agility and Design, Education & Collaboration, as well as Team Stories, Customers, & Products.

We would like to extend our thanks to the authors of the papers in this volume for their contributions to the field of Agile software development. We would also like to thank the reviewers for their time and effort in ensuring the quality of the papers - as well as all the speakers, sponsors, shepherds, chairs, and volunteers. Finally, we would like to express our gratitude to the XP Conference Steering Committee and the Agile Alliance for their ongoing support.

We hope that the research proceedings of the XP 2023 conference will serve as a valuable resource for researchers, practitioners, and students interested in Agile software development.

June 2023

Christoph J. Stettina  
Juan Garbajosa  
Philippe Kruchten

# Organization

## Conference Co-chairs

Wouter Lagerweij  
Suzanne Lagerweij

Lagerweij Consultancy, The Netherlands  
Lagerweij Consultancy, The Netherlands

## Program Co-chairs

Christoph J. Stettina

Leiden University/Centre for Innovation,  
The Netherlands

Juan Garbajosa

Universidad Politécnica de Madrid, Spain

## Publication Chairs

Philippe Kruchten  
Peggy Gregory

University of British Columbia, Canada  
University of Glasgow, UK

## Program Committee

Pekka Abrahamsson  
Steve Adolph  
Ademar Aguiar  
Scott Ambler  
Craig Anslow  
Leonor Barocca  
Hubert Baumeister  
Jan Bosch  
Frank Buschmann  
Daniela N. Cruzes  
Torgeir Dingsøy

Tampere University, Finland  
Cprime, Canada  
University of Porto, Portugal  
SA+A, Canada  
Victoria University of Wellington, New Zealand  
Open University, UK  
Technical University of Denmark, Denmark  
Chalmers University of Technology, Sweden  
Siemens AG, Germany  
NTNU, Norway  
Norwegian University of Science and Technology,  
Norway  
Kinneret Academic College, Israel  
Independent, Germany  
Blekinge Institute of Technology, Sweden  
Free University of Bozen-Bolzano, Italy

Yael Dubinsky  
Jutta Eckstein  
Henry Edison  
Ilenia Fronza

Juan Garbajosa	Universidad Politécnica de Madrid, Spain
Alfredo Goldman	University of São Paulo, Brazil
Peggy Gregory	University of Glasgow, UK
Eduardo Guerra	Free University of Bozen-Bolzano, Italy
Tomas Gustavsson	Karlstads Universitet, Sweden
Helena Holmström Olsson	University of Malmö, Sweden
Kiyoshi Honda	Osaka Institute of Technology, Japan
Fabio Kon	University of São Paulo, Brazil
Philippe Kruchten	University of British Columbia, Canada
Thomas Kude	University of Bamberg, Germany
Marco Kuhrmann	Reutlingen University, Germany
Casper Lassenius	Aalto University, Finland
Ville Leppänen	University of Turku, Finland
Lech Madeyski	Wroclaw University of Science and Technology, Poland
Sabrina Marczak	PUCRS, Brazil
Antonio Martini	University of Oslo, Norway
Frank Maurer	University of Calgary, Canada
Tommi Mikkonen	University of Helsinki, Finland
Alok Mishra	Atılım University, Turkey
Nils Brede Moe	SINTEF, Norway
Parastoo Mohagheghi	Norwegian Labour and Welfare Administration, Norway
Jürgen Münch	Reutlingen University, Germany
Anh Nguyen Duc	University College of Southeast Norway, Norway
Tyron Offerman	Leiden University, The Netherlands
Maria Paasivaara	LUT University & Aalto University, Finland
Cécile Péraire	Carnegie Mellon University, USA
Rafael Prikladnicki	PUCRS, Brazil
Adam Przybyłek	Gdansk University of Technology, Poland
Pilar Rodríguez	Universidad Politécnica de Madrid, Spain
Helen Sharp	Open University, UK
Darja Šmite	Blekinge Institute of Technology, Sweden
Simone Spiegler	Monash University, Australia
Christoph J. Stettina	Leiden University/Centre for Innovation, The Netherlands
Viktoria Stray	University of Oslo, Norway
John F. Tripp	Clemson University, USA
Rini Van Solingen	Delft University of Technology, The Netherlands
Joost Visser	Leiden University, The Netherlands
Stefan Wagner	University of Stuttgart, Germany
Xiaofeng Wang	Free University of Bozen-Bolzano, Italy



Hironori Washizaki  
Agustin Yague

Waseda University, Japan  
Universidad Politécnica de Madrid, Spain

## **Additional Reviewers**

Halimeh Agh  
Pavel Nedvědický  
Ana Moises de Souza  
Eva Zimmermann

University of Stuttgart, Germany  
University of Stuttgart, Germany  
Norwegian University of Science and Technology,  
Norway  
University of Stuttgart, Germany

## **Steering Committee**

Hubert Baumeister  
François Coallier  
Jutta Eckstein  
Teresa Foster  
Juan Garbajosa (Chair)  
Peggy Gregory  
Wouter Lagerweij  
Casper Lassenius  
Maria Paasivaara  
Viktoria Stray

Technical University of Denmark, Denmark  
École de technologie supérieure, Canada  
Independent, Germany  
Agile Alliance, USA  
Universidad Politécnica de Madrid, Spain  
University of Glasgow, UK  
Lagerweij Consultancy, The Netherlands  
Aalto University, Finland  
LUT University & Aalto University, Finland  
University of Oslo, Norway

## **Sponsoring Organization**

Teresa Foster

Agile Alliance, USA

# Contents

## Agile Practices

Integrating Issue Management Systems of Independently Developed Software Components .....	3
<i>Sandro Speth, Uwe Breitenbücher, Niklas Krieger, Pia Wippermann, and Steffen Becker</i>	
A Novel Technique to Assess Agile Systems for Stability .....	20
<i>Robert Healy, Tapajit Dey, Kieran Conboy, and Brian Fitzgerald</i>	
Overcoming Challenges of Virtual Scrum Teams: Lessons Learned Through an Action Research Study .....	34
<i>Jedrzej Bablo, Bartosz Marcinkowski, and Adam Przybylek</i>	
Waste Self-reporting for Software Development Productivity Improvement .....	50
<i>Marc Sallin, Martin Kropp, Craig Anslow, and Robert Biddle</i>	
A Lean Approach of Managing Technical Debt in Agile Software Projects – A Proposal and Empirical Evaluation .....	67
<i>Abdullah Aldaej, Anh Nguyen-Duc, and Varun Gupta</i>	
An Empirical Study About the Instability and Uncertainty of Non-functional Requirements .....	77
<i>Luiz Viviani, Eduardo Guerra, Jorge Melegati, and Xiaofeng Wang</i>	

## Agile in the Large

Striving for Freedom in a Large-Scale Agile Environment with an Entrepreneurial Mindset of a Product Owner .....	97
<i>Piret Niva, Maria Paasivaara, and Sami Hyrynsalmi</i>	
Sustaining Agility: Organizational Change, Factors and Theoretical Lenses .....	115
<i>Leonor Barroca, Helen Sharp, Advait Deshpande, Peggy Gregory, and Stavros Papadeas</i>	
The Role of Responsiveness to Change in Large Onboarding Campaigns .....	132
<i>Darja Smite and Nils Brede Moe</i>	

Organizational Conflicts in the Adoption of Continuous Software Engineering ..... 149  
*Eriks Klotins and Elliot Talbert-Goldstein*

Data-Driven Development in Public Sector: How Agile Product Teams Maneuver Data Privacy Regulations ..... 165  
*Astri Barbala, Tor Sporseem, and Viktoria Stray*

**Short Paper**






Real-Life Water-Scrum-Fall: Insights from Large Companies in Czech Republic ..... 183  
*Michaela Křivánková and Daniel Remta*

**Author Index** ..... 193

# **Agile Practices**



# Integrating Issue Management Systems of Independently Developed Software Components

Sandro Speth<sup>1</sup>, Uwe Breitenbücher<sup>2</sup>, Niklas Krieger<sup>1</sup>,  
Pia Wippermann<sup>1</sup>, and Steffen Becker<sup>1</sup>

<sup>1</sup> Institute of Software Engineering, University of Stuttgart, Stuttgart, Germany  
{Sandro.Speth,Niklas.Krieger,Pia.Wippermann,  
Steffen.Becker}@iste.uni-stuttgart.de

<sup>2</sup> Herman Hollerith Zentrum, Reutlingen University, Reutlingen, Germany  
uwe.breitenbuecher@reutlingen-university.de

**Abstract.** Modern component-based architectural styles, e.g., microservices, enable developing the components independently from each other. However, this independence can result in problems when it comes to managing issues, such as bugs, as developer teams can freely choose their technology stacks, such as issue management systems (IMSs), e.g., Jira, GitHub, or Redmine. In the case of a microservice architecture, if an issue of a downstream microservice depends on an issue of an upstream microservice, this must be both identified and communicated, and the downstream service's issues should link to its causing issue. However, agile project management today requires efficient communication, which is why more and more teams are communicating through comments in the issues themselves. Unfortunately, IMSs are not integrated with each other, thus, semantically linking these issues is not supported, and identifying such issue dependencies from different IMSs is time-consuming and requires manual searching in multiple IMS technologies. This results in many context switches and prevents developers from being focused and getting things done. Therefore, in this paper, we present a concept for seamlessly integrating different IMS technologies into each other and providing a better architectural context. The concept is based on augmenting the websites of issue management systems through a browser extension. We validate the approach with a prototypical implementation for the Chrome browser. For evaluation, we conducted expert interviews, which approved that the presented approach provides significant advantages for managing issues of agile microservice architectures.

**Keywords:** Microservices · Issue management · Service engineering · Component-based architectures · Gropius · Browser extension

## 1 Introduction

The component-based architectural style, e.g., microservices, gained much attention since it enables the development of individual software components independently from each other and composing them systematically to new systems [1]. Developing components independently has significant advantages, e.g.,

microservices can be implemented in different programming languages by different teams [2]. However, this independence can also result in drawbacks when it comes to managing issues [3], such as bug reports or feature requests. For example, in microservice architectures, we can observe that often different issue management systems are used for different microservices. Moreover, as soon as an architecture includes 3<sup>rd</sup>-party services, such as SaaS offerings, or open-source components, these, of course, have their own issue management systems, and it is clear that not all services are developed by the same team. Therefore, it is very unlikely that all components of a larger system use the same issue management system. Hence, different teams might use different *issue management systems* (IMSs), e.g., Jira, GitHub, or Redmine [4]. In the following, we use microservices as an example of independently developed components to make the problem clear. For example, suppose a malfunction, anomaly, or failure of a *Checkout* microservice actually results from a bug in an invoked *Payment* microservice, e.g., missing parameters through a change violating the services contract [5]. Figure 1 depicts such a scenario. In this case, the malfunction of the *Checkout* microservice should be reported as an issue relating to its causing bug of the *Payment* microservice, hence, allowing the developers of the *Checkout* service to track the bug report’s status of the *Payment* service. Furthermore, the *Checkout* service’s team should communicate the issue to the *Payment* service’s developer team to resolve the issue. However, documenting and tracking such *cross-component issues* is challenging [3], primarily since no issue management system supports explicit relations to other issues managed by another issue management system as different IMSs are not integrated with each other. Thus, if issues of microservices are managed by different tools, e.g., one microservice is managed in Jira, the other in Redmine, it is not supported by any IMS to semantically relate the issues with each other, which is often required to understand the dependencies of issue propagations, e.g., because of cascading failures, and how issues need to be solved. Furthermore, developers often communicate via the issue’s comments and keeping track of the related issues results in switching between the different issue management systems. This leads to many context switches and, thus, ineffective communication and preventing the developers from focused work. However, especially in modern agile processes like Scrum, “getting things done” is a central value that is thereby hindered. Furthermore, it is required to describe the dependency relation between the two issues while putting them in their correct architectural context to prevent the developers of *Checkout* microservice from trying to fix the malfunction separately, which is problematic since another microservice causes it. Therefore, we require to bypass the limitation of IMS boundaries and integrate their issues into each other, which leads to our first research question:

**RQ1:** “How could different issue management systems such as Jira and GitHub be seamlessly integrated to avoid context switches through additional tooling?”

In addition, developers need an efficient solution to discover dependencies of their own issue to issues in other issue management systems, as these are often

hidden in current systems only via natural language in the comments. This leads us to our second research question:

**RQ2:** “How could developers in their own issue management system quickly identify dependencies of their own issues on issues of other issue management systems?”

For this reason, in this paper, we propose a concept that makes issues from different issue management systems available in each other in an IMS-independent way. The concept is based on seamlessly augmenting the websites of issue management systems such as Jira, Redmine, or GitHub via Browser extensions. In doing so, we use Gropius, which we developed in previous work [6, 7], in the extensions to synchronize issues and enable Gropius features, such as semantic links in the IMS, that do not otherwise exist. Gropius is a standalone IMS platform that enables issue management across the boundaries of every single software component and their IMSs while putting the issues in their correct architectural context. However, while Gropius offers a top-down view of the entire architecture and their issues as a separate tool, developers still require a seamless developer-specific view from inside a component in their usual issue management systems without switching between IMSs. Therefore, our concept maps the top-down view of the architecture to a developer-specific view of the architecture from within a component. This hides unnecessary or disruptive information from the developer and puts the focus on the developer’s own component and the upstream or downstream components from it. Thus, if certain issues of a component are shown, for example, in GitHub, associated issues that might be managed by other issue management systems such as Jira are directly embedded in the website of GitHub, which enables quickly recognizing dependencies and navigating to other issues. Developers should therefore get their work done faster and communicate more efficiently via the dependent issues’ comments with other teams. We validated the practical feasibility of this *Gropius Browser Extension* concept by a prototypical implementation for the Chrome browser and GitHub as an exemplary issue management system. Moreover, we evaluated the concept and the prototype through expert interviews. The experts approved that the approach provides significant advantages for managing issues of individual services in a component-based architecture, e.g., independently developed microservices.

## 2 Fundamentals of Issue Management and Gropius

Today’s issue management systems (IMS), such as Jira or Redmine, allow developers to document, manage and track issues, e.g., bug reports or feature requests, for their projects, discuss the issues’ context with other developers, and collaborate on them. Often, architectural design decisions are also discussed in issues as they both influence each other. Therefore, issue management systems often act as the main tool to coordinate their work and become increasingly central to every software development process as a hub for communication

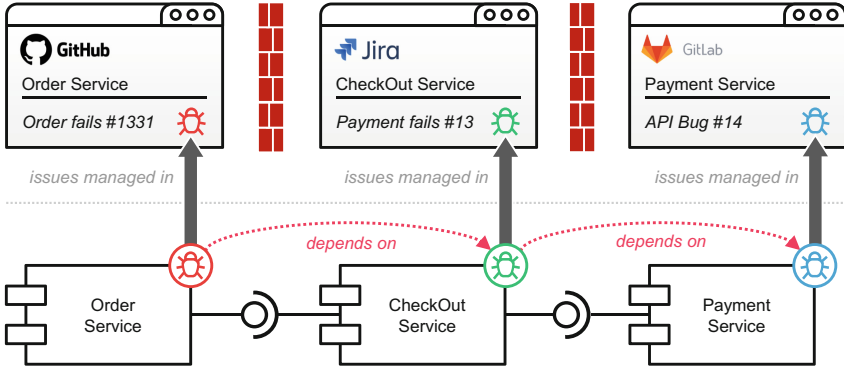


Fig. 1. Motivating scenario showing how bugs propagate through the architecture.

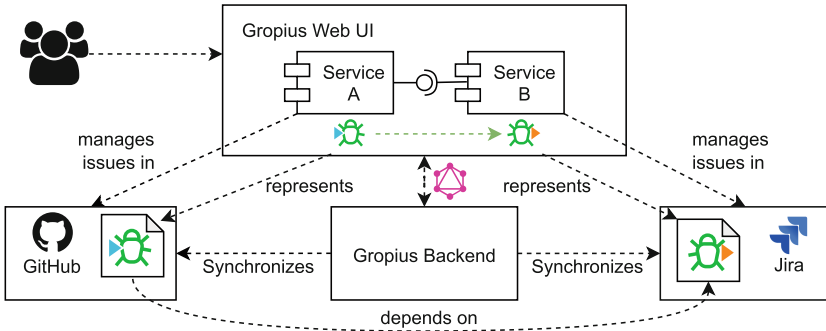
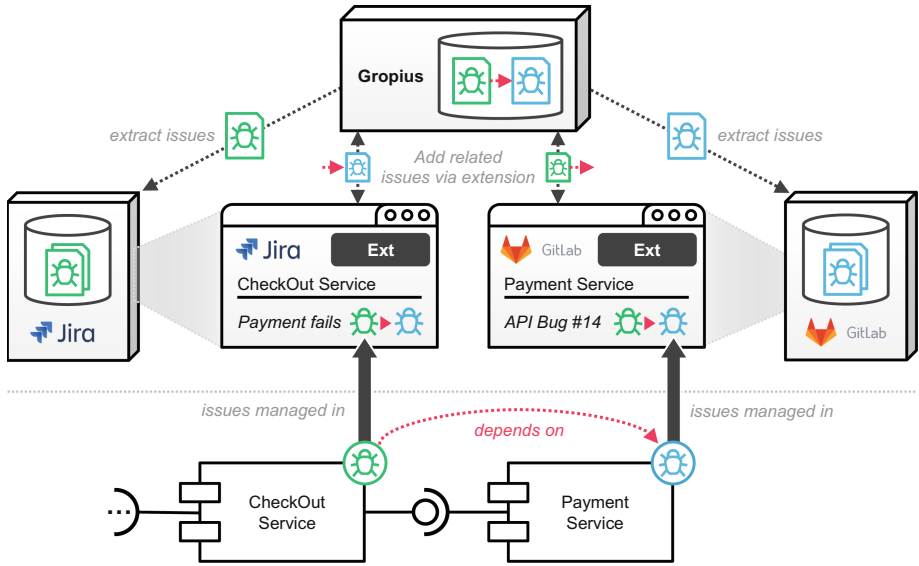


Fig. 2. Overview of the Gropius concept and architecture.

and collaboration [8]. Furthermore, issue management systems offer the possibility for distributed negotiations required to resolve the issue [9]. However, in modern software development, individually developed components typically use their independent issue management systems. Therefore, linking an issue from one IMS to another systems’ issue, e.g., for a dependency relation, is not directly possible as traditional IMSs are restricted to their component’s boundaries. Nevertheless, related issues should be linked amongst each other [8] in order to provide all helpful information required to resolve the issue and increase its quality [10]. Gropius attempts to remove this limitation and facilitate (1) synchronization of issues across different issue management systems, and (2) relations between those cross-component issues. In Gropius, each issue can be associated with one or more components that together form the architecture of a software system. Such a software system is represented in a project in Gropius, i.e., a *Gropius Project*. In general, Gropius supports various kinds of components, e.g., microservices, libraries, and infrastructure. One goal of Gropius is to effectively represent the impact of issues on the architecture and issue propagation, as well as architectural dependencies between the components. For this reason, in the *Gropius Web UI*, components of such a Gropius project are modeled in a UML





**Fig. 3.** Overview and architecture of the Gropius Browser Extension concept.

component diagram-like graph. Stakeholders with an overview of the entire software system, e.g., a software architect, can create representations of components and their provided interfaces in Gropius and connect them to model the architecture. Furthermore, each component in Gropius is either Gropius internal or requires a URL to its actual IMS. Each component is attached with the issues that affect that component. Through Gropius, developers can create relations between issues of different components which are added to the issues. These relations are graphically represented by arrows and collectively depicted in the issue view metadata. To enable these cross-component issue features while leveraging the issues of the actual IMSs, Gropius acts as a wrapper over traditional IMSs, such as GitHub or Jira, as depicted in Fig. 2 and offers a GraphQL API to clients. Through specific adapters for the different IMSs, Gropius synchronizes issues between the systems. Furthermore, Gropius keeps a copy of the issues, especially for data that is not supported by the actual IMSs, such as cross-IMS issue relations.

### 3 The Gropius Browser Extension Concept

This section presents the conceptual idea of the *Gropius Browser Extension (GBE)* for integrating different issue management systems used by independently developed software components. First, we give a brief overview of the concept in Sect. 3.1. Afterwards, we explain the concept with regard to elicited requirements in detail. Please note that while we use microservices as a common example for independently developed software components, the GBE concept is not restricted

to microservices but also enables other independently developed and includable software projects, e.g., libraries or infrastructure components such as Kubernetes and Docker. Therefore, we use the general term “*component*” for the remainder.

### 3.1 General Idea and Objectives

Figure 3 shows the general idea based on the motivating scenario. The main objective is to enable *cross-component issue management across different issue management systems in a seamless manner* using their standard websites in a traditional way. The websites get seamlessly enriched by the extension with information about related issues of dependent components in the architecture, which are possibly managed by other issue management systems. This integration concept avoids two drawbacks of existing issue management approaches: (i) First, relations between issues managed by different IMSs are typically documented using textual comments, which provides no systematic means and hinders “browsing” them. (ii) Second, additional tooling, such as the Gropius Web App, that enables cross-component issue management is not required for creating, relating, and managing issues, thus, reducing context switches of developers.

### 3.2 Overview and Architecture of the GBE

The Gropius Browser Extension is a plugin that runs in a web browser and automatically manipulates the HTML DOM of shown websites, in our case, the websites of issue management systems such as Jira or Redmine. The extension connects to the Gropius backend, which contains information about issues in different IMSs and also contains dependencies between them as well as other metadata [6, 7]. Thus, the Gropius backend acts here as a *normalized store* for issues from different IMSs, which are automatically extracted and synchronized by Gropius in a certain time interval.

The GBE has access to the DOM of the currently shown page of the issue management system, from which it retrieves relevant context information, e.g., which issue is currently shown. Using this information, the GBE calls the Gropius backend to fetch cross-component issue features for the current issue, e.g., related issues and their respective components. This information is additionally injected into the issue management system’s website by manipulating its HTML DOM, hence, enabling Gropius functionality directly in the component’s issue management system’s website following its look and feel. Thus, this provides a seamless integration of the Gropius features into the regular work of developers as no additional website or tool needs to be used to see issue relations, issues of other components, etc. Especially, content based on different issue management systems can be integrated into the developer’s used issue management system.

In the following, we concretely describe the Gropius features that are integrated using the GBE into the standard website of the issue management systems. Please note that we describe in this section only the concepts and provide details on the technical implementation in Sect. 4, which describes our open-source prototype of the Gropius Browser Extension.

### 3.3 Gropius Features Integrated in the Browser Extension

In this section, we describe the Gropius features and functionalities that are integrated by the Gropius Browser Extension into the standard websites of issue management systems and how they are realized in the presented architecture.

**Browser Extension Feature 1: Creating Gropius Project.** Before developers can use any Gropius features in the used IMS, they must create a project in Gropius, add relevant components to this project, and specify the respective issue management systems. This functionality is supported directly by the GBE as otherwise the external Gropius Web App needs to be used, which would lead to context-switches. Therefore, we integrated a feature to create a Gropius project or to add an IMS's project, e.g., a GitHub repository, to an existing project. This way, developers can add microservices, libraries, and other components to their Gropius project when browsing through the respective project of the IMS used. Please note that the way this feature is shown and how it is triggered, e.g., through a button, is IMS-specific. Therefore, the browser extension will look and feel slightly different for GitHub than for Jira and other systems.

**Browser Extension Feature 2: Issue Dependencies and Architectural Context.** One main objective of the Gropius Browser Extension is to enrich an issue currently shown in an issue management system's website with additional information regarding its dependencies to other issues. Especially if the related issues belong to another component, the extension must provide the architectural context in an understandable fashion in order to enable the developer to quickly identify the impact and dependencies of the issue to other components and their issues. Therefore, the GBE injects two kinds of additional information to the issue view of an IMS: (1) the related issues of the same component and of other components, and (2) an excerpt of the architecture graph that shows the component of this issue and also all other components having issues to which the currently shown issue relates to.

For the first kind of injected information, i.e., the list of related issues of the same or other components, the Gropius Browser Extension shows for the currently shown issue which components and other issues the shown issue directly affects or is otherwise related to. To easily create relations between the current issue viewed to other issues of the same or a different component, the Gropius Browser Extension enables to add such relations directly in the issue view page of the issue management system, thus, no further tooling is required. For example, if developers create an issue for the own component and know that this issue actually results from another component's issue, i.e., has a dependency to it, they are interested in directly adding the relation to the other issue in order to specify the complete dependencies. Without the Gropius Browser Extension, developers would have to add a text comment including the URL to the related issue of a possibly different issue management system and write an explanation which describes the semantics of the relation. Of course, text-based issue relations are

only a workaround that tries to compensate this obviously missing feature to link issues in different systems. On the other side, using the Gropius Browser Extension, this problem is solved since relations to issues of the same component or other components can be directly added in the issue’s view page including a precise specification of the semantics of the relation, e.g., that the issue *depends on* an issue of another component in another IMS.

In general, adding such relations are especially complex for issue management systems that already allow semantic issue relations, for example GitLab’s *Linked issues* or Redmine’s *Related issues*. While such issue management systems allow relating issues within the same issue management system provider, relating issues to other providers, e.g., issues of a GitHub repository, is impossible. Therefore, the GBE enables integrating relations to issues of different issue management system providers in the currently shown issue view.

The second kind of information the Gropius Browser Extension adds to an issue’s view is an excerpt of the overall architecture of the system that shows (a) the component of the currently shown issue as well as (b) all other components having issues to which the shown issue relates. Thus, this provides a visual overview of the architectural context and enables developers to quickly understand the impacts and causes that the shown issue has, i.e., all dependencies.

**Browser Extension Feature 3: Entire Architecture of the System.** The Gropius Browser Extension aims at enabling developers a quick overview of issues and their relations to other issues of possibly other components in the architecture. Therefore, the previously introduced Browser Extension Feature 2 enables to show an architectural excerpt that provides an overview of the directly related issues and their respective components. However, this excerpt only shows a part of the system’s architecture and often developers need to see the entire architecture of the system in which the current component is used. Therefore, the GBE enables to load the entire architecture of a system that includes all components, their issues, and their dependencies into the currently used issue management system’s website. Thus, developers can browse the entire architecture and directly jump to the IMSs of other components by clicking on the component in the graph or on one of its issues. In the latter case, a new tab opens the other IMS and shows the clicked issue. This enables browsing through the entire architecture of the system directly in the IMS and avoids that another tool needs to be used by developers to inspect the overall system architecture.

Please note that a certain component and its issues can be part of multiple different systems. For example, a component such as the Payment Service might be used in our webshop motivation example but also in a car rental application or other applications that require payment functionality. In this case, a certain component would be part of multiple different system architectures which are managed in Gropius as individual Gropius projects. Thus, one Gropius project represents one system, describing its architecture, and listing all components with the corresponding issues. Therefore, the Gropius Browser Extension enables

switching between Gropius projects in the issue management systems website and then shows the architecture and all components of current selected project.

**Browser Extension Feature 4: Create Issues for Other Components Managed in Other Issue Management Systems.** In many situations, a bug in one component results from a bug in an invoked component. For example, in the motivating scenario, the bug in the Payment Service’s API leads to an issue in the CheckOut Service that depends on this API. Thus, if a developer of the CheckOut Service detects that the checkout functionality does not work correctly, the developer needs to report this as an issue for the CheckOut Service. However, if the developer directly understands that the root cause of this bug is the broken API of the Payment Service, maybe as a result of a failure root cause analysis, also a bug should be created for the Payment Service and the two issues should be linked, i.e., that the checkout bug results from the payment’s API bug. Therefore, the Gropius Browser Extension supports creating issues also for other components of the architecture directly in the issue’s view page of the issue management system, even if the current IMS is different from the IMS of the other component. Based on this feature, developers do not require to open the other issue management system to report the causing issue. This is a significant advantage since such a context switch requires the developers to first orient themselves in the other issue management system.

## 4 Prototypical Validation

To validate the practical feasibility of the presented concept, we implemented a prototype of the Gropius Browser Extension for the Google Chrome Browser. The implementation is open source available<sup>1</sup>. Section 4.1 describes the technical design of the prototype. Moreover, Sect. 4.2 shows how the implemented browser extension for Chrome supports GitHub, i.e., how the extension interacts with the Web UI of GitHub and how issues from other issue management systems are embedded.

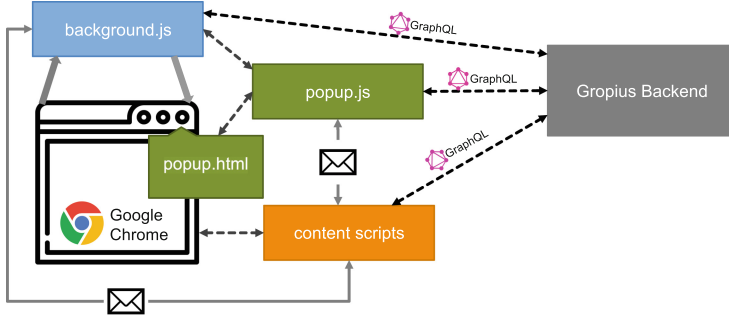
### 4.1 Technical Design of the Gropius Browser Extension for Chrome

The architecture of our Gropius Browser Extension is based on the *web extensions framework*, which is an official W3C standard cross-browser architecture<sup>2</sup> and supported by all popular desktop web browsers except for Safari.

Figure 4 depicts our prototype’s architecture for the Google Chrome browser. Following the web extension framework, our prototype consists of three components (1) *background pages*, (2) *UI pages*, and (3) *content scripts*. The extension executes background pages as soon as loaded. Especially code that maintains long-term state and operations, which should be independent of any website’s

<sup>1</sup> <https://doi.org/10.5281/zenodo.6810943>.

<sup>2</sup> <https://browserext.github.io/browserext/#availability-csp-content>.



**Fig. 4.** Architecture of the Gropius Browser Extension prototype for Google Chrome.

or browser window’s lifetime, must be added as background pages. In the context of our extension, the background scripts decide whether one of the content scripts should be active, i.e., the content script for a GitHub issue view, if the developer has opened the GitHub page. When clicking on the extension popup button in Chrome’s upper right part, it offers UI elements and options for login to Gropius to the user via the UI pages component, i.e., *popup.js* and *popup.html*. While the background and UI pages can access the browser’s API, they cannot interact with the issue management system’s website. In contrast, the content scripts interact with the issue management system’s website, however, having only limited access to the browser’s API. Therefore, content scripts run in the context of the issue management system’s website, and the background and UI pages run in the context of our GBE. Content scripts enable the way websites are displayed to the user and their functionalities to be manipulated. In our context, this means adding UI elements related to Gropius to the issue management system’s website, thus, enhancing the issue management system’s website with additional Gropius functionalities. For example, a button to add a new related issue in the issue’s view, which opens a dialogue to add the relation, is injected here. The communication between content scripts, background, and UI pages works via messages. Therefore, components listen to other components’ messages and respond via the same channel. The extension interacts with the Gropius backend via its provided GraphQL API to add or retrieve architectural and Gropius project information, issues, and issue relations.

## 4.2 Implementation for GitHub and Case Study

We implemented our prototype in Vue.js, a progressive JavaScript framework for web applications. For our prototype, we especially use the plugin “vue-cli-plugin-browser-extension”, a third-party library for Vue that supports creating extensions for popular browsers. For our prototype, we focus on Google Chrome as web browser and the issue management system of GitHub. In the following, we describe the prototype based on an extended version of the example scenario

## Failed to checkout a new order #1

Edit

New issue

Open

opened this issue 13 hours ago · 0 comments

**Gropius Component Graph**

```

graph LR
    Order-Service -- REST-API --> CheckOut-Service
    CheckOut-Service -- REST-API --> Payment-Service
  
```

Active Gropius Projects: Webshop

Related Issues:

- Payment fails if credit card holder's na...  
Component(s): Payment-Service
- Cannot process order with German uml...  
Component(s): Order-Service

Assignees: No one—assign yourself

Labels: bug

Projects: None yet

commented 13 hours ago (Owner)

Brief: When calling the API endpoint to check out a new order and provide the order ID, and selected payment information, the API returns an HTTP status code 400 Bad Request. Additionally, the log contains error messages regarding a parsing error of the credit card holder's name, if the name contains special characters.

Steps to reproduce:

**Fig. 5.** Browser screenshot of the Gropius Browser Extension prototype.

described in Sect. 1, i.e., the webshop application consisting of the (1) Order Service, the (2) CheckOut Service, which is the service our development team focuses on, and the (3) Payment Service.

The Checkout Service consumes the Payment Service’s API to handle the payment of orders. The Checkout Service’s issues are managed in GitHub and the other services in different IMSs. The prototype allows adding a GitHub repository as a component to an existing Gropius project (cf. Feature 1). To add a GitHub repository which is opened in the web browser to a Gropius project, the extension injects a button in the sidebar of the repositories “Code” tab. After modelling the provided interfaces and architectural dependencies in the Gropius Web App, developers can add relations between issues of the CheckOut Service and issues of the other components (cf. Feature 2). Figure 5 depicts an example issue for the CheckOut Service, which describes a bug regarding checking out an order. On top of the issue’s description, the GBE injects an architecture graph to show the issue’s architectural context (cf. Feature 2 & 3). Our prototype injects two additional metadata in the issue view’s sidebar above the assignees. The “Active Gropius Projects” part allows switching the active Gropius project in which the CheckOut service is included. In the example, the Gropius project “Webshop” is selected as active. Therefore, the extension shows in the “Related Issues” part related issues and their components in the context of this project (cf. Feature 2). The icons next to the related issue’s titles indicate whether the currently viewed issue depends on the related issue or causes it. Additionally, the “Related Issues” part allows developers to add new relations to issues of

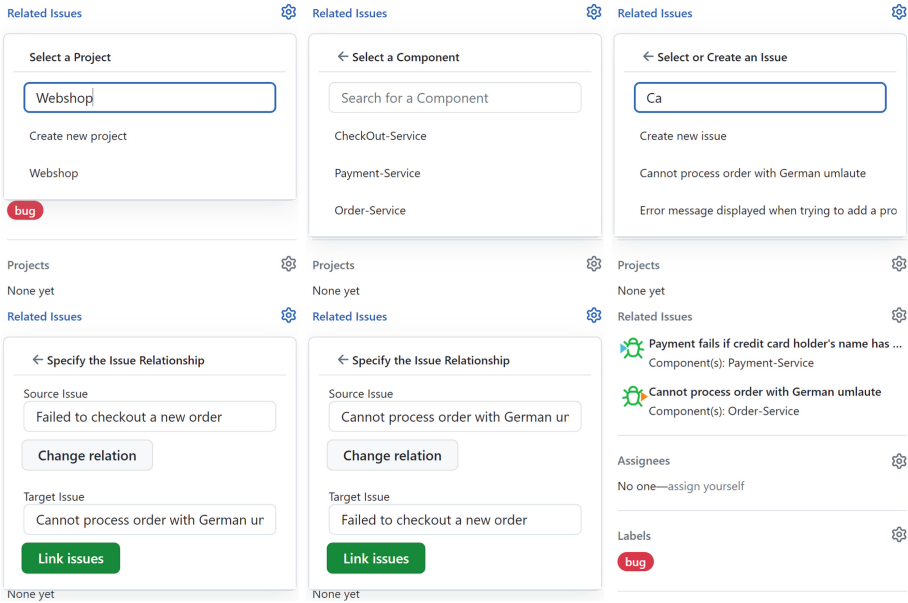


Fig. 6. Steps for adding a related issue.

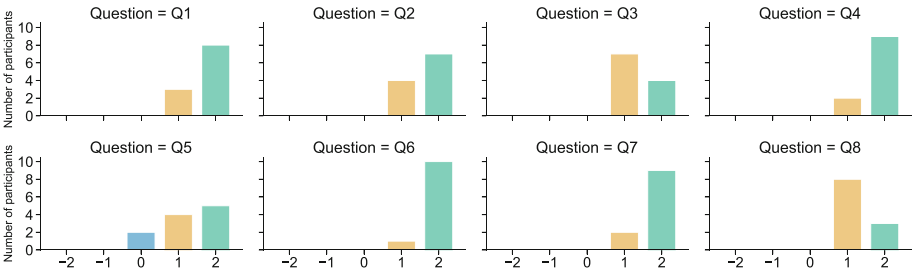


Fig. 7. Results of the evaluation for each question.

the same or different component or remove existing relations. As depicted in Fig. 6, when adding new relations, the extension opens a dialogue frame where a developer has first to select the component of the related issue and then select the issue which should be related. Furthermore, instead of selecting an existing issue, a developer can create new issues for the chosen component directly in the issue’s view (cf. Feature 4).

## 5 Evaluation

We evaluated our concept based on a *Goal Question Metric* approach and a prototype version of our software. We formulated the goal “enabling integrated context-aware issue relationship management for independently managed



*software systems*". Based on this, we derived questions evaluating the general concept and how our prototype fulfills the goal. To answer the questions, we conducted an expert survey. As the rating scale for the questions, we used five interval points where the endpoints are named to match associated questions. Therefore, the scale ranges from  $-2$  over  $0$  to  $2$ , where  $-2$  means an entirely negative evaluation,  $0$  is the neutral element, and  $2$  means an entirely positive evaluation. Since our concept aims especially to improve the developers' issue management across multiple independently managed components, we consulted eleven experienced industrial software developers from different small-sized to big-sized companies in Germany and Canada. The participants worked between 3 and 15 years in developing component-based systems, during the last years also focusing on microservices. Most of their roles are developer and software architect. Furthermore, while developing, their components interacted with components developed by other teams. We conducted the expert survey by hosting interviews in which we showed the prototype version of the GBE and an example architecture with issues to the participants. Afterwards, we provided them with our questionnaire.

## 5.1 Results of the Expert Survey

This section describes the questions and the results of the survey.

- (Q1) *"How helpful is the main feature of the Gropius Browser Extension to see related issues of other components?"* The scale ranges here from *not helpful at all* ( $-2$ ) to *very helpful* ( $+2$ ). In total, eight experts (73%) rated (Q1) with  $+2$  and three (27%) rated it with  $+1$ , which means that they evaluated to see related issues of other components in a concise way as very helpful.
- (Q2) *"How big do you estimate the time savings enabled by the Gropius Browser Extension to find related issues of other components (compared to the common approach without the extension)?"* The interviewees were asked to rate this with a value ranging from *very low* ( $-2$ ) to *very high* ( $+2$ ). Seven of the experts (64%) rated  $+2$ , four experts (36%) rated  $+1$ . Thus, developers estimate that the extension may help saving time for finding related issues.
- (Q3) *"How important do you consider the presented concept of the Gropius Browser Extension for the development process of software and its operation?"* The interviewees were asked to rate this with a value ranging from *not important at all* ( $-2$ ) to *very important* ( $+2$ ). For (Q3), four experts (36%) rated a  $+2$  while seven experts (64%) stated the importance of our concept for the development process with a  $+1$ . Signals that our concept is generally important but not significantly required.
- (Q4) *"How well do you find the Gropius Browser Extension integrated into the look and feel of GitHub?"* The interviewees were asked to rate this with a value ranging from *not good at all* ( $-2$ ) to *very good* ( $+2$ ). Overall, nine experts (82%) evaluated the integrated look and feel of our prototype with

+2, and two experts (18%) stated a +1, which strongly indicates that our prototype seamlessly integrates into GitHub.

- (Q5) “*How high do you rate the chance that the Gropius Browser Extension will be accepted in the industry?*” The interviewees were asked to rate this with a value ranging from *very low* (−2) to *very high* (+2). The acceptance of our concept within the industry was evaluated as relatively high. However, the results are well distributed between +2 and 0, i.e. five times (45%) +2, four times (36%) +1, and two times (18%) 0. Therefore, not every expert was convinced of acceptance in the industry.
- (Q6) “*How important do you consider the concept of cross-component issues for the software engineering process of complex systems that consist of several independent components?*” The interviewees were asked to rate this with a value ranging from *not important at all* (−2) to *very important* (+2). For (Q6), ten experts (91%) rated the concept of cross-component issues as very important, i.e., appointing a value of +2, and one expert (9%) esteemed it as important with a value of +1.
- (Q7) “*How quickly can you recognize that the issue currently viewed has dependencies on issues of other components?*” The interviewees were asked to rate this with a value ranging from *very slow* (−2) to *very fast* (+2). With nine experts (82%) rated (Q2) with +2 and two experts (18%) rated a +1, the average of +1.82 means that the experts could recognize dependencies on issues other component’s issues very fast, thus, supporting the usability.
- (Q8) “*How user friendly is editing and adding related issues?*” The interviewees were asked to rate this with a value ranging from *not user friendly at all* (−2) to *very user friendly* (+2). Three experts (27%) judged the user friendliness for editing and adding related issues with a +2 and eight (73%) with a +1. Therefore, the average of +1.27 supports the user friendliness.

## 5.2 Summary of the Evaluation and Threats to Validity

To summarize, the feedback regarding our concept was very positive. Therefore, our expert survey shows that a problem in managing cross-component issues in complex software systems, such microservices, is seen, supporting some of the challenges outlined by Mahmood et al. [3]. Often, ratings of +1 and below were justified by the fact that there are more important tasks, especially regarding the entire development process and operation. Additionally, our prototypical implementation was rated very good. In particular, the immersive integration and design to GitHub’s UI was frequently emphasized in debriefings. While our concept provides one possible answer for our first research question, especially the answers to question 7 support that integrating the dependency graph into the traditional IMS’s UI and adding relations to other issues in a semantic clear way enables efficient identification of the own issue’s dependencies, thus, answering our second research question.

Please note that since we conducted an expert survey, the given answers depend on personal opinions. Thus, the results are subjective, resulting in a

threat to internal validity and external validity regarding how representative our surveyed group is [11]. Furthermore, there is a threat to construct validity if the experts misunderstood some questions. However, these threats were solved by allowing the interviewees to ask questions that were answered.

## 6 Related Work

There are several attempts for cross-component issue management. Various Redmine and Jira forums<sup>3</sup> discuss how issues can be related to multiple IMS projects. Proposed solutions are always limited to one IMS provider and are usually plugins for the respective IMS providers instead of a provider-independent browser extension. Issue tracking across the boundaries of an IMS provider is not enabled, and is, therefore, not practical for our use case where components are developed independently and thus manage their issues in different IMS providers, e.g., GitHub and Jira. Especially for Jira and GitHub, plugins exist that enable approaches that support multiple IMS projects from the same provider. Synchronization of issues across multiple Jira projects is enabled by the Jira plugin *Backbone Issue Sync*<sup>4</sup> and the *Multi Project Picker*<sup>5</sup> Jira plugin removes the restriction that an issue can only belong to one Jira project. For GitHub, there is the plugin ZenHub<sup>6</sup> which offers additional project management functionalities. With ZenHub, issues are not restricted to one GitHub repository anymore. However, ZenHub does not support linking GitHub issues to issues of other IMS providers. Additionally, there are browser extensions for issue management in Jira which aim at saving time, e.g., JIRA Assistant, Zephyr, JIRA Template Injector, and Google to Jira. However, similar to the plugins above, none of the extensions enables issue relations across different IMS projects or providers. We followed with GropiusVSC, an IDE extension for Gropius, a similar approach as we to reduce context-switches and improve a developer's productivity [12]. GropiusVSC offers a developer for a selected component of a Gropius project the list of issues and an issue view. The issue view shows the issues title, description, and related issues with their components. By clicking on a related issue of another component, this component is automatically selected as active component, and GropiusVSC shows the issue's view. While GropiusVSC is helpful during development time, a relevant part of a developer's week consists of meetings, e.g., Sprint Planning, or other occasions where developers work within the actual issue tackers' web pages, and, therefore, not sufficient alone. Hence, our browser extension helps developers by integrating context-aware issue management across a component's boundary in the component-specific IMS.

<sup>3</sup> <http://bit.ly/3EdNU0g>, <https://bit.ly/Iuk3NrP>, and <https://bit.ly/3IKKv2n>.

<sup>4</sup> <https://www.k15t.de/software/backbone-issue-sync-for-jira>.

<sup>5</sup> <http://bit.ly/3xqofh5>.

<sup>6</sup> <https://www.zenhub.com/>.

## 7 Conclusion

The central objective of this work was that developers of a component could manage cross-component issues in the context of the issue management system they use. We achieved this by the Gropius Browser Extension concept that seamlessly integrates various issue management systems via the Gropius backend. Our prototype has shown that the concept can be practically realized while the conducted expert interviews in the evaluation confirm the relevance of the approach. The main advantages of the approach are the reduction of developers' context switches and the avoidance of a separate tool in the development process to manage issues in different management systems. In future work, we will extend our prototype for all common issue management systems, such as Jira.

## References

1. Szyperski, C., Gruntz, D., Murer, S.: *Component Software: Beyond Object-oriented Programming*. Pearson Education (2002)
2. Nygard, M.: *Release It!: Design and Deploy Production-Ready Software*. Pragmatic Bookshelf (2007)
3. Mahmood, S., Niazi, M., Hussain, A.: Identifying the challenges for managing component-based development in global software development: preliminary results. In: *Science and Information Conference (SAI)*. IEEE 2015, pp. 933–938 (2015)
4. Speth, S.: Semi-automated cross-component issue management and impact analysis. In: *Proceedings of 2021 36th IEEE/ACM International Conference on Automated Software Engineering*, IEEE, pp. 1090–1094 (November 2021)
5. Ramírez, F., Mera-Gómez, C., Bahsoon, R., Zhang, Y.: An empirical study on microservice software development. *SESoS/WDES* **2021**, 16–23 (2021)
6. Speth, Sandro, Breitenbücher, Uwe, Becker, Steffen: Gropius — a tool for managing cross-component issues. In: Muccini, Henry, Avgeriou, Paris, Buhnova, Barbora, Camara, Javier, Caporuscio, Mauro, Franzago, Mirco, Koziolok, Anne, Scandurra, Patrizia, Trubiani, Catia, Weyns, Danny, Zdun, Uwe (eds.) *ECSA 2020. CCIS*, vol. 1269, pp. 82–94. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-59155-7\\_7](https://doi.org/10.1007/978-3-030-59155-7_7)
7. Speth, S., Becker, S., Breitenbücher, U.: Cross-component issue metamodel and modelling language. In: *Proceedings of the 11th International Conference on Cloud Computing and Services Science*, SciTePress, pp. 304–311 (May 2021)
8. Bertram, D., Volda, A., Greenberg, S., Walker, R.: Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In: *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, pp. 291–300 (2010)
9. Sandusky, R.J., Gasser, L.: Negotiation and the Coordination of Information and Activity in Distributed Software Problem Management. In: *Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work*, ACM, pp. 187–196 (2005)
10. Bettenburg, N., et al.: What makes a good bug report? In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. *SIGSOFT 2008/FSE-16*, pp. 308–318. ACM (2008)

11. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. *Emp. Softw. Eng.* **14**(2), 131–164 (2008)
12. Speth, S., Krieger, N., Breitenbücher, U., Becker, S.: Gropius-VSC: IDE support for cross-component issue management. In: *Companion Proceedings of the 15th European Conference on Software Architecture*, CEUR (October 2021)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# A Novel Technique to Assess Agile Systems for Stability

Robert Healy<sup>1</sup> , Tapajit Dey<sup>2</sup> , Kieran Conboy<sup>3</sup> , and Brian Fitzgerald<sup>2</sup> 

<sup>1</sup> Intive Ireland, 6th Floor O'Connell Bridge House, Dublin 2, Ireland  
rob.healy@intive.com

<sup>2</sup> Lero – The SFI Research Centre for Software, University of Limerick, Limerick, Ireland

<sup>3</sup> School of Business and Economics, University of Galway, Galway, Ireland

**Abstract.** Agile systems, like the Kanban and Scrum frameworks, are built on assumptions of sustainability and stability, however, there is little empirical evidence on whether such systems are stable in practice or not. Therefore, in this study we aim to inspect the stability of Agile systems by leveraging the concept of stability described in Queueing Theory. We define a novel metric, the Stability Metric, as a way of assessing queueing systems, especially Agile systems. We inspect 926 Jira projects in 14 organizations with over 1.6 million product backlog items using this metric. The analysis showed that 72.89% of these Jira projects were not stable and stable systems, on average, had product backlog sizes 10 times shorter than unstable ones. These results suggest that while the goal of Agile is to create a sustainable, stable way of working, this is not guaranteed, and a better understanding of systems and queues may be required to help design, create, coach, and maintain optimal Agile systems.

**Keywords:** Agile · Queueing Theory · Stability · Jira · Backlogs

## 1 Introduction

The United Nations define sustainability as “meeting the needs of the present without compromising the ability of future generations to meet their own needs” [1]. Agile software development has existed at least since the 2001 Agile Manifesto, with many of the frameworks predating the manifesto itself [2]. One of the principles of Agile is that “Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely” [3]. Two of the most popular Agile frameworks are Scrum and Kanban [2]. Both frameworks can be modelled as queueing systems. In queueing systems, sustainability occurs under conditions of mathematical stability [4]. In this paper, we present a novel approach by leveraging the concept of stability to assess if Agile systems are stable and sustainable in practice as is commonly assumed.

To measure the stability of Agile systems we used queueing theory to derive a new metric, the Stability Metric (SM) to classify the performance of queues. We applied this metric to 926 collections of Jira Projects (JPs), of Product Backlog Items (PBIs) from the

more than 2.5 million records in the Public Jira Dataset [5]. The distribution of Stability Metric across all JPs and the relationship between Stability Metric and product backlog size and inter-service arrival time was extracted and analyzed.

The rest of the paper is organized as follows: In Sect. 2, we discuss and briefly present the background and related concepts. We describe the research approach in Sect. 3 and the results in Sect. 4. We provide further discussion about the results and its implications in Sect. 5. Finally, we describe the limitations to our study in Sect. 6 and offer conclusions in Sect. 7.

## 2 Background

### 2.1 Queuing Theory for Stable Queuing Systems

Queuing systems have been researched since Erlang’s work on telecommunications in the early 20<sup>th</sup> century [6]. A queue forms in a service when demand for that service exceeds supply [6] and one or more interconnected queues form a queuing system [4]. Each queue can be stable, unstable, or marginally stable [4]. A queue is considered stable when the Markov chain of all possible queuing states is ergodic in nature [4]. This means that a stable queue must include the possibility of occasionally having no items in it. Erlang defined the traffic intensity,  $\rho$  as the dimensionless ratio of the average arrival rate,  $\lambda$ , to the average service rate,  $\mu$ , where both  $\lambda$  and  $\mu$  are measured in items per unit time as shown in Eq. 1 [6].

$$\rho = \frac{\lambda}{\mu} \quad (1)$$

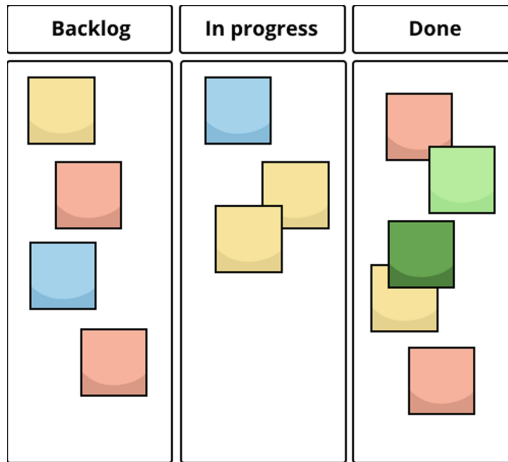
In this case, the queue can be seen to be stable when the traffic intensity is less than one, or when the service rate is greater than the arrival rate. In telephony systems, this allows for the service rate to be fixed and the system is controlled by altering the arrival rate to ensure that the system is stable. From Jackson’s Theorem for networks of queues we know that a network of queues will be stable only when all the sub-queues within that network are stable [4]. For this research we will model Agile frameworks as one or more queues.

### 2.2 Modelling Agile Frameworks as Systems of Queues

Kupainen *et al.* [7] conducted a systematic literature review of 774 papers to identify key metrics in use by Agile teams for planning, progress tracking, software quality measurement, fixing software process problems, and motivating people. One of the metrics for progress tracking they identified is Work-in-Progress (WIP) limits. WIP extends from Little’s Law – an observation that in stable queues the amount of time an item spends in a queue,  $W$ , is equal to the average number of items in the queue,  $L$ , divided by the average arrival rate of items,  $\lambda$  [8]. Therefore, in theory, it is possible to control the throughput of a stable Agile queue by simply enforcing a limit on the number of WIP items by using Eq. 2 [8].

$$W = \frac{L}{\lambda} \quad (2)$$

The Kanban framework is already constructed around the principles of queueing theory [8]. A simple Kanban system, such as that shown in Fig. 1, is a queue with one or more servers. The “Backlog” column represents the queue, the “In progress” column represents the servers and the “Done” column represents items exiting the queue. Unlike traditional queueing systems, the order of a Kanban backlog tends not to be a simple First-In First-Out (FIFO) or Last-In First-Out (LIFO) queue and instead tends to be dynamically re-ordered. Also, many real Kanban systems tend to have multiple *columns* and *swimlanes*. A *column* usually represents an upstream or downstream service such as “development” and “testing”. However, a column can also be a separate queue if items are placed there without any action being performed on them. Examples of this include “Ready for testing” or “Waiting / Blocked”. A *swimlane* is a horizontal division across a Kanban board that often represents a separate class of service – e.g., a priority item. Each swimlane may be considered as an independent queue.



**Fig. 1.** Sample Kanban board

The Scrum guide, on the other hand, defines two queues – a primary “product backlog” that is managed and dynamically prioritized by the Product Owner and a “sprint backlog” that is selected by the team at the sprint planning event, with items chosen from the product backlog based on available capacity [9]. As a result of how items move from product backlog to sprint backlog, standard Scrum systems tend to be more batched than simple queues. However, the overall system still can be modelled as a simple queue if the arrival and service rates are stochastic in nature and can be modelled using a Poisson and exponential distribution respectively [4]. In practical terms, a stable Scrum system, like a stable Kanban system and like other similar stable Agile queueing systems, has the advantage of being highly predictable. This happens when the team are, on average, more than capable of delivering the work at least as fast as it is being requested of them.



### 3 Research Approach

With the theoretical foundation of the Queuing Theory in mind, the primary research question we are addressing in this paper is: ***RQ: Are Agile systems stable from a queuing perspective?*** To answer this question, we use a metric derived from the Queuing theory called the Stability Metric, as described below, and analyze a Public Jira dataset [5] containing 16 public Jira repositories involving 1822 Jira projects to determine if those systems were stable.

#### 3.1 Introducing the Stability Metric (SM)

For Agile queues, the arrival rate of items into a product backlog is often not within the control of the team, or even the organization if customer-reported bugs are included. However, the service rates can be controlled through team design, scaling multiple teams, and training and coaching individual team members. Although queuing theory is known in Agile literature and metrics such as cycle time and WIP limits exist [7, 8], typical measures of queues as systems have not been used. We define a new term – the “Stability Metric (SM)”,  $\psi$ , that is the inverse of the traffic intensity and is shown in Eq. 3.

$$\psi = \frac{\mu}{\lambda} \quad (3)$$

The Stability Metric ranges from zero to infinity; values from zero to lower than one indicates an unstable queue system, a value of one indicates a marginally stable system, and values above one indicates stability. The Stability Metric has the advantage over its predecessor, the “traffic intensity” metric, that the items that are within control are focused on the numerator rather than the denominator so a positive change in ability results in the Stability Metric growing. Unlike existing popular Agile metrics such as “velocity” or “cycle time” [7], the Stability Metric is not purely a lagging metric. Instead, we believe it may be used as both a diagnostic metric to understand why a system is performing/underperforming and as a design tool to help organize teams and Agile systems to manage predicted workloads effectively. We use this metric to help us address our research question.

#### 3.2 Analyzing the Public Jira Dataset

Past studies of Agile systems have focused on contextual realism with case studies of real software engineering teams, although these studies are typically not grounded in theory [10]. Montgomery *et al.* [5] curated and published a dataset of the contents of the 16 public issue tracking systems with 1822 Jira projects and 2.7 million Product Backlog Items (PBIs) all using the Atlassian Jira issue tracking tool, which they suggest is the leading issue-tracking tool for Agile systems. We decided to analyze this dataset to make the results as generalizable as possible. This dataset, however, is not very clean for our purposes, i.e., it is hard to determine if individual Jira projects use any standard Agile framework, or the degree to which Jira projects can be mapped to a framework. While some parts of the dataset suggest an Agile implementation (e.g., the existence of User Stories which is an XP concept that is often used in Scrum [2]), there are other parts of

the dataset that suggest that it could be used by ITIL/traditional waterfall-type projects (for instance, “Change Requests”) or helpdesk-type work such as “Support Request”. We assumed that most or all the 2.7 million PBIs will be part of one or more queues and that some of these queues are part of an Agile framework. Using data from the dataset it is possible to calculate the arrival and service rates of each issue and calculate the stability.

To analyze the Public Jira Dataset, we first downloaded it from its public repository and, following instructions from Montgomery *et al.* [5], restored it to a MongoDB database. The dataset is organized into a set of 16 “repos” where each repo is the extracted Jira information from one of the issue tracking systems. Using a Python script, we extracted the following fields from each repo: *Issue ID*, *Project Name*, *Issue Type*, *Subtask Boolean*, *Assignee ID*, *Created Date/Time*, *Resolution Date/Time*, *Status Name*. We subsequently collated this data into a comma separated value (CSV) file for each organization for each month between January 2002 and January 2022. These CSV files were then combined into a Microsoft Excel file for all issues for each Jira repo.

The Jira repos are divided into “Projects” where Atlassian advise all issues related to a product should be assigned to a Jira Project (JP) [11]. Only JPs with more than 30 issues were considered, this was to reduce skew caused by new/inactive/abandoned JPs and allow the assumption of a normal distribution around the calculated mean as per the central limit theorem. With that filtering criterion in mind, we removed the *SecondLife* and *Mindville* repos as they had fewer than 30 issues each. For the remaining repos, all the Epic issue types and Subtask issue types were filtered out to ensure the work was approximately similarly sized. Epic issue types are used in Jira as “parents” of other ticket types [12] and remain unresolved longer than other ticket types, potentially skewing service rates. On the other hand, Subtask ticket types are used in Jira as “children” of standard ticket types [13] and, as such, tend to be resolved in shorter periods, also potentially skewing the data.

When team completes a PBI, it receives a resolution. Resolution types can be configured per JP, but some default types exist [14]. There were 81 separate resolution types across the JPs in the dataset. Of these only six denoted the delivery of a successful piece of work. Table 1 lists them below. The other seventy-six resolution types denoted abandoning the queue before the team completed the PBI. We removed all PBIs with these resolutions also.

**Table 1.** Top 6 successful resolution types.

Resolution	Total PBIs	Percentage of total resolved PBIs
Fixed	968080	41.9%
Done	371312	16.1%
Resolved	2904	0.1%
Deployed	136	0.0%
Fixed-Verified	116	0.0%
Delivered	102	0.0%

This resulted in 926 JPs with 1,633,166 PBIs between 2002 and 2022. 1,295,002 PBIs were successfully resolved by teams during that the sampling period. For each JP, we calculated the arrival rate and service rate. We calculated the arrival rate,  $\lambda$ , for each system, as per Eq. 4. We calculated the service rate,  $\mu$  using Eq. 5. We calculated the Stability Metric,  $\psi$ , as shown in Eq. 3, from these and the results were grouped into Unstable ( $\psi < 1$ ), Stable ( $\psi > 1$ ), and Marginally Stable ( $\psi = 1$ ).

$$\lambda = \frac{\#tickets\ created\ that\ are\ unresolved\ or\ resolved\ successfully}{(datetime\ of\ last\ ticket\ created - datetime\ of\ first\ ticket\ created)} \quad (4)$$

$$\mu = \frac{\#tickets\ created\ that\ are\ resolved\ successfully}{(datetime\ of\ last\ ticket\ resolved - datetime\ of\ first\ ticket\ resolved)} \quad (5)$$

As stated in Sect. 2.1, a requirement of stability is for a queue to be a Markovian chain with the property of ergodicity. In such a scenario, the size of the queue will eventually and temporarily drop to zero. The dataset allowed us to evaluate how big queue/product backlog was for each JP at the point where the data was captured. We calculated the product backlog,  $L_{JP}$ , by taking the total PBI's that had arrived,  $A_{JP}$ , and subtracting the total PBI's that has been resolved,  $Z_{JP}$ , as per Eq. 6. We plotted these in relation to the backlog size.

$$L_{JP} = A_{JP} - Z_{JP} \quad (6)$$

Finally, we devised a proxy test for Agility. Equation 7 shows the calculation for the average inter-service time for each JP. Both Scrum and Kanban advocate for discretization of work, breaking work into small parts and monitoring progress daily [2]. In Scrum all PBI's must fit into a timeboxed sprint that is commonly two weeks in duration but can be any period up to a month in length [9]. Large epics and very small sub-tasks had been filtered at an earlier stage, so this analysis acts as a gauge of how many JPs may have contained Agile systems by measuring how many could not have fit in a Scrum system. Systems with PBIs with durations of weeks, months or longer could indicate a phased based "waterfall" approach.

$$t_{JP} = \frac{1}{\mu_{JP}} \quad (7)$$

## 4 Results

In this section, we discuss the results of the analysis outlined above. First, we present the stability distributions of all 926 relevant JPs in the Public Jira Dataset. Then we present the relationship between the stability and backlog size. Finally, we show the data on average inter-service time and demonstrate its relationship to stability.

### 4.1 Stability of JPs

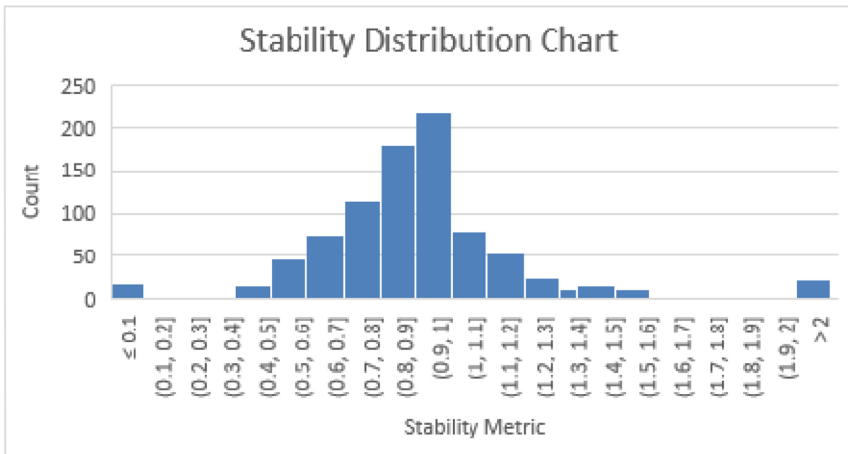
Table 2 shows that nearly three-quarters, 72.89%, of JPs are unstable from a queueing perspective, with around one quarter, 24.92%, appearing to be stable and the remaining

**Table 2.** Stability Metrics of PBIs

Stability	Stability metric, $\psi$	JP count	Percentage
Unstable	0.77	675	72.89%
Marginal	1.00	20	2.16%
Stable	1.80	231	24.92%

2.16% marginally stable. Figure 2 shows the distribution of stability across all JPs with a clear tendency for JPs to cluster around marginal stability and a tendency for most to be slightly unstable with 54.05% of all JPs being in the range 0.7–0.99.

Figure 2 also shows that there are outliers on both the upper and lower ends. At the lower end, 2.04% of all JPs have an arrival rate more than ten times faster than the service rate. For every PBI delivered by the people working on such a system, ten new PBIs arrived at the same time, on average. At the upper end of the scale, 4.39% of JPs result in workers involved there having nothing to do on the JP more than half of the time, on average.



**Fig. 2.** Distribution of the Stability Metric across 926 JPs

Table 3 shows the proportion of stable and marginally stable JPs as a share of all JPs. This shows that none of organizations have consistently stable systems. There does not appear to be a relationship between the number of JPs per organization/repo and the percentage of stable systems. However, 86% of organizations have at least one stable JP.

### 4.2 Stability and Backlog Size

When Montgomery *et al.* captured the Public Jira Dataset in 2022, 338,164 PBIs were unresolved across 887 JPs. Table 4 shows the relationship between each level of stability

**Table 3.** Stable and marginally stable JPs for all Repos.

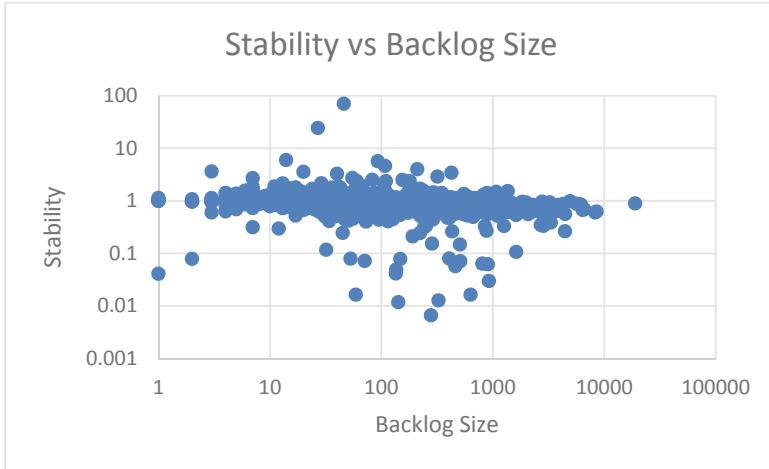
Repos	Number of stable/marginally stable JPs	Count of JPs	Percentage
JFrog	0	6	0.0%
Sonatype	0	2	0.0%
Jira	2	26	7.7%
Mojang	1	8	12.5%
Qt	3	17	17.7%
JiraEcosystem	10	55	18.2%
Sakai	4	20	20.0%
MariaDB	2	9	22.2%
Apache	125	486	25.7%
MongoDB	8	27	29.6%
RedHat	65	198	32.8%
Hyperledger	7	18	38.9%
Spring	22	52	42.3%
IntelDAOS	1	2	50.0%
<b>All JPs</b>	<b>250</b>	<b>926</b>	<b>27.0%</b>

and the corresponding backlog size for all 926 JPs. It shows that the biggest backlog size appears when the system is unstable – this makes sense as, by definition, the system is unable to meet the demands being placed on it. The smallest backlog sizes occur when the system is marginally stable. Again, this is logical, since for those systems, the time spent waiting for a new piece of work to arrive is eliminated – there is minimal waste, but the system is not overloaded. Systems with high stability may wait to accumulate PBIs before addressing at high speed.

**Table 4.** Stability metrics and backlog sizes of PBIs.

Stability	Mean stability metric, $\psi$	Mean backlog size	JP count
Unstable	0.77	455.39	675
Marginal	1.00	80.25	20
Stable	1.80	126.55	231

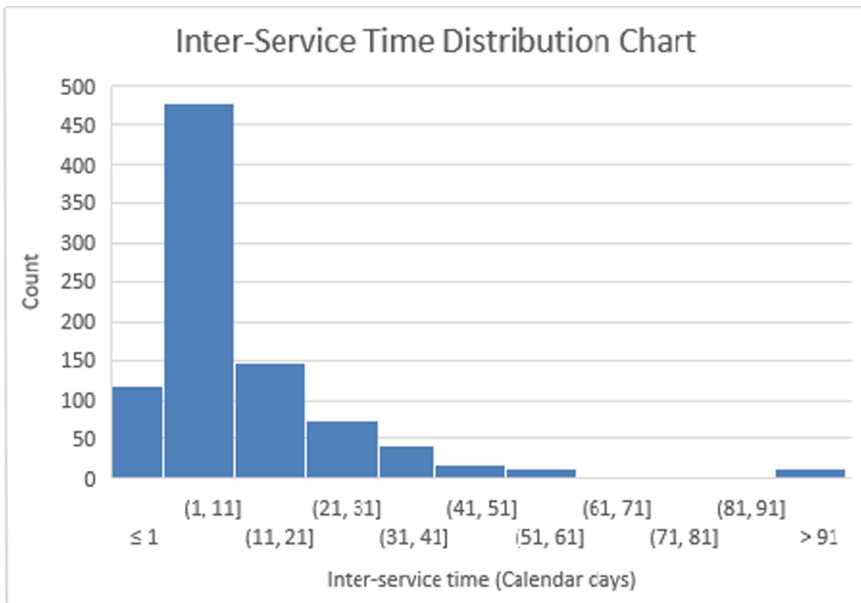
Figure 3 expands on Table 2 by illustrating the relationship between stability and backlog sizes. A point to note is the significant ranges in backlog sizes which vary between 0 and 18,956. Of these, the largest backlogs are an order of magnitude larger than marginally stable or stable queues. 71% of all stable and marginally stable systems have backlogs between 0 and 100 items in them. By contrast, 49% of all unstable systems have backlogs of one hundred or more.



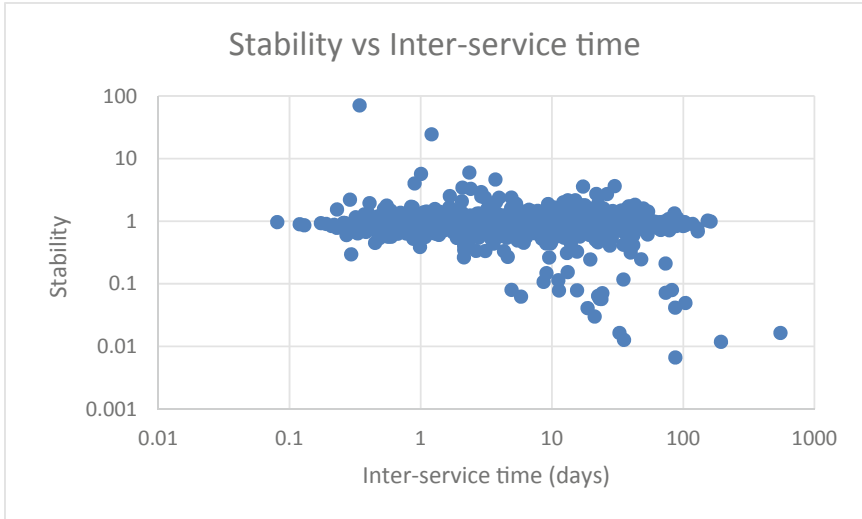
**Fig. 3.** Backlog size vs stability of all JPs, plotted on logarithmic axes.

### 4.3 Stability and Inter-service Times

Figure 4 shows the distribution of average inter-service times for all resolved PBIs. These were calculated using Equation 7. 64% of PBIs are resolved within eleven calendar days and that 88% are resolved within 31 calendar days. This indicates that some PBI's may have been used in an Agile system.



**Fig. 4.** Distribution of inter-service time



**Fig. 5.** Inter-service time vs stability of all JPs, plotted on logarithmic axes.

Figure 5 shows the relationship between inter-service time and stability. The outliers on the lower right in this plot demonstrate that very unstable systems tend to have high service times while the outliers on the upper left show that systems with very high stability tend to have very fast service times. However, perhaps the most interesting information on the plot is the broad range of inter-service times over which systems tend to cluster around marginal stability. Over 33% of all JPs have stability in the range of 0.9 to 1.1. Of these, the service rates range from 0.08 days between PBIs resolved on average to 161.58 days between PBIs resolved on average. This suggests that a wide array of systems with different arrival rates, people, tools, technologies are all independently all adapting processes to attempt to stabilize their systems. These results will be discussed further in the Discussion section.

## 5 Discussion

The analysis of the 926 JPs of the Public Jira Dataset illustrates that it is possible to extract anonymous data from Jira projects and process it using a novel measure, the Stability Metric to be able to infer the stability of the system. We illustrated the relationship to backlog size for a large historic dataset with over 1.6 million items over a 20-year period. The results show a strong tendency for the system to be unstable with 72.86% showing this. These unstable systems have, by definition, growing backlogs and we also find that they tend to have larger backlogs overall. Our results show that there is a discrepancy between the assumed stability of Agile systems and the reality of how systems operate in practice and seems to indicate that the assumption of sustainability associated with the Agile process may not hold good in practice due to the systems' lack of stability.

However, over half of all JPs are nearly stable, with 57.6% of all JPs are in the range 0.8–1.2 so future research may investigate whether it is possible, or advisable, for

systems to be adapted so they become more stable in nature. Of the 14 organizations whose data was analyzed, all but two had some systems either stable or marginally stable, ( $\psi \geq 1$ ). This means that although all organizations had some unstable systems, most could make some systems stable. Further investigation of real teams using the stability metric can help identify which techniques help teams achieve stable ways of working, and whether these correlate with other metrics or measures of satisfaction.

An interesting result is the percentage of stable systems in the dataset – 27% of all JPs analyzed were stable or marginally stable. From a practical perspective the tendency towards a Stability Metric value of close to 1 makes sense as organizations that achieve this can deliver work as needed, without having frequent idle resources. This finding was supported by the analysis of the relationship between average backlog sizes and the stability metric, with a tendency for significantly larger backlog sizes to correspond to unstable systems. Whilst this is suggestive of a causal relationship between backlog size and stability, this also will need to be tested with actual teams.

Another effect worth considering in further research is the impact the statistical necessity of having no backlog has on system stability. A team with nothing to do is likely to be assigned to other work by the management since their concern of using the available manpower optimally conflicts with the conditions of having a mathematically stable system where a team having no work is expected, and even required, to happen periodically. In Scrum, a high-performing team with no carryover of work achieves this to a degree every sprint but a Kanban team who have a continuous flow of work may never eliminate work-in-progress and achieve a stable system. This calls into question the applicability of Little's Law which requires a stable system to be applicable. Further research is needed to analyze real organizations to assess whether high utilization rates is a higher priority than system performance, because this may suggest a business incentive for not striving to achieve sustainable work systems.

Despite most JPs appearing to be close to stable where  $\psi \approx 1$ , significant outliers do exist where JPs had high levels of instability  $\psi < 0.1$  and stability  $\psi > 2$ . Both outliers are worth considering as neither are sustainable for the people performing the work nor the organizations in which these systems exist. Where  $\psi < 0.1$ , the average arrival rate is more than 10 times the average service rate. This is likely to be a stressful situation, as demand outstrips supply and difficult choices will need to be continually made as to which is the next most important piece of work. For systems where  $\psi > 2$  this means that the people involved in performing the work have nothing to do for more than half the time on average. Neither outlier looks sustainable from a business or human perspective – the business would under-perform compared to competitors and the developers might suffer from burnout or boredom. Future research is needed to determine if these stability metrics exist in Agile systems, as it is not possible to determine the precise conditions from the data, but if substantiated, it would strongly reject a hypothesis of sustainable ways of working as neither sponsor, developer nor user could be reasonably expected to sustain unstable ways of working indefinitely.

## 5.1 Potential Application of this Metric

The Stability Metric is still under investigation and, as Sect. 6 describes, there remain several limitations in the investigations to date. It may be useful to discuss potential



applications. This metric is adapted from existing measures from queueing theory but is novel to analysis of Agile systems. It is intended as a diagnostic tool to help predictability. For all Agile systems it can help diagnose if the team is being under-loaded or over-loaded and the degree to which this is occurring. This, in turn may help in overall organizational design and establishing the appropriate number of teams for the volume of work required.

On a more operational level, by shifting the duration over which the data is used a comparison can easily be made between stability over a longer period and stability in the recent past. This will provide information to the team and stakeholders on the impact of continuous improvement initiatives. In Kanban systems, the Stability Metric can be used to determine if the system is stable and if applying Little's Law and limiting WIP is a valid approach. It could be used on an overall system process flow and on any sub-system within that flow to quantify and control process bottlenecks. In Scrum, it could be an improvement on the current practice of "yesterday's weather" to help teams identify the target velocity required per sprint to achieve a stable product backlog based on longer term "climatic" patterns of arrival and service rates. However, before we get too excited about potential applications, we must recognize this research is ongoing and list the limitations of the work to date.

## 6 Limitations

The use of a large dataset that crosses many organizations for a prolonged period offers very good generalizability of findings, but it is at the payoff of contextual realism. The main limitation of this work is that it is not known with 100% certainty if an organization we studied used an Agile framework. The analysis of service rates shown in Fig. 4 suggests that the majority PBIs were discretized to fit within short cycles used in Scrum and Kanban. Also, the use of Jira itself is suggestive of Scrum or Kanban approaches as these are standard frameworks within the tool [16].

Another limitation of this study was the use of Jira Projects, JPs, as a collection of PBIs that represent a queuing system. Jira can use a JP for this purpose but may also slice the collections of PBIs within and across JPs to create queues for teams. Without the specific details of what queue or queues are in use in a given organization it is not possible to be certain of the stability of each. We have relied on Jackson's Theorem for queueing networks which dictates that a network of queues can only be stable if all sub-queues in the network are stable [4]. This suggests that for JPs that displayed stability, any sub-queues in that JP must have also been stable. The situation is more complicated for unstable queueing networks; it is possible for a network to be unstable as long as one or more of the sub-queues are unstable. Further investigation of real teams will be needed to determine the prevalence of instability.

Methodologically, one limitation may arise in the fact that we used averages across the entire JP. This potentially ignores temporal effects. A queue is a dynamic system and may be temporarily stable or unstable. The degree to which stability fluctuates and root causes should be investigated further, probably with real teams.

Finally, the study assumed that the data provided were accurate. Since the creation and update of PBIs in Jira is a human activity there is likely to be variation in the accuracy

of the data. For example, we assumed the work arrived in the queue at the point where the PBI was created but it is plausible that sometimes a piece of work could be discussed long before it is logged in Jira. Similarly, a ticket can be resolved only to find out the resolution was insufficient, and more work is needed. Jira captures only the date of first resolution [14]. We removed subtasks, Epics, and certain resolution types but these may have been misclassified by the original user who was more interested in getting their work done than data integrity. Further investigations will be required into the Public Jira Dataset to analyze the accuracy of the data captured.

## 7 Conclusion

The research question considered whether Agile systems are stable from queueing perspective. A novel metric was developed to test this hypothesis and a large historic dataset was used to test the Stability Metric. While it is not known how many of the Jira Projects sampled used Agile frameworks such as Scrum and Kanban, it is likely that many did. Based on this assumption, the data presented show that systems are often unstable with large and growing product backlogs. A potential cause of this is likely service rates that are too slow for the individual queue. Further research is required to investigate in more detail but based on this analysis it appears that Agile software development systems are neither inherently stable nor sustainable from a human or business perspective but can, under certain conditions, be made so.

## References

1. United Nations:Sustainability. <https://www.un.org/en/academic-impact/sustainability>. Accessed 5 Apr 2023
2. Measey, P., et al.: *Agile Foundations: Principles, Practices and Frameworks*, pp. 125–162. BCS, Swindon (2015)
3. Fowler, M., Highsmith, J.: The agile manifesto. *Softw. Dev.* 9(8), 28–35 (2001)
4. Bose, S.: *An Introduction to Queueing Systems*, pp. 17–22. Kluwer Academic/Plenum, New York (2002)
5. Montgomery, L., Luders, C., Maalej, W.: An alternative issue tracking dataset of public Jira repositories. In: *Proceedings of the 19th International Conference on Mining Software Repositories*, pp. 73–77 (2022). <https://doi.org/10.1145/3524842.3528486>
6. Kleinrock, L.: *Queueing Systems*, vol. 1, pp. 17–19. John Wiley & Sons, New York (1975)
7. Kupiainen, E., Mäntylä, M., Itkonen, J.: Using metrics in agile and lean software development – a systematic literature review of industrial studies. *Inf. Softw. Technol.* 62, 143–163 (2015). <https://doi.org/10.1016/j.infsof.2015.02.005>
8. Vacanti, D.: *Actionable Agile Metrics for Predictability*, pp. 41–54. Actionable Agile Press, LeanPub (2015)
9. Fuior, F.: Key elements for the success of the most popular Agile methods. *Revista Română de Informatică și Automatică* 29(4), 7–16, (2019). <https://doi.org/10.33436/v29i4y201901>
10. Conboy, K., Fitzgerald, B.: Toward a conceptual framework of agile methods: a study of agility in different disciplines. In: *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research*, pp. 37–44 (2004). <https://doi.org/10.1145/1029997.1030005>
11. What is a Jira Software Project. <https://support.atlassian.com/jira-software-cloud/docs/what-is-a-jira-software-project/>. Accessed 5 Apr 2023

12. What is an epic. <https://support.atlassian.com/jira-software-cloud/docs/what-is-an-epic/>. Accessed 5 Apr 2023
13. Create an issue and a sub-task. <https://support.atlassian.com/jira-software-cloud/docs/create-an-issue-and-a-sub-task/>. Accessed 5 Apr 2023
14. Defining resolution field values. <https://confluence.atlassian.com/adminjiraserver/defining-resolution-field-values-938847105.html/>. Accessed 5 Apr 2023



**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# Overcoming Challenges of Virtual Scrum Teams: Lessons Learned Through an Action Research Study

Jedrzej Bablo<sup>1</sup>, Bartosz Marcinkowski<sup>2</sup>(✉) , and Adam Przybyłek<sup>3</sup> 

<sup>1</sup> Lufthansa Systems Poland, Grunwaldzka 415, 80-309 Gdansk, Poland

<sup>2</sup> University of Gdansk, Bazynskiego 8, 80-309 Gdansk, Poland

bartosz.marcinkowski@ug.edu.pl

<sup>3</sup> Gdansk University of Technology, Narutowicza 11/12, 80-233 Gdansk, Poland

**Abstract.** After the COVID-19 breakout, agile teams found themselves in situations that “pure agilists” and textbooks on agile methods had preferred to ignore. Whereas agile mindsets helped them to quickly shift to remote work, mere virtualization of agile practices often proved insufficient, and several challenges emerged. This paper reports on an Action Research project carried out in Lufthansa Systems Poland with the aim of (1) revisiting their ad-hoc actions to adapt to remote work; and (2) elaborating systematic solutions to maintain efficiency in such a setting. With our assistance, the participating teams found measures to mitigate issues posed by the new work environment. They devised an inter-team communication model to improve the effectiveness of information exchange that had declined in the absence of spontaneous, face-to-face communication. Moreover, they employed several other mitigation strategies, including working at least one day per week in the office, keeping webcams on during online meetings, and recapping meetings at the end of a session. Our study largely supports previous findings indicating that Scrum can be effectively applied beyond its comfort zone but also suggests that for adaptations to be successful and comprehensive, they should be developed in a structured manner.

**Keywords:** Remote work · Method tailoring · Agile Software development · Adaptation · Teamwork · Collaboration

## 1 Introduction

When COVID-19 swept over the world, many businesses were suddenly disrupted and forced to make rapid changes to the workplace and work processes [1–3]. During these turbulent times, organizational agility not only proved to be useful but also often made the difference between success and failure. Not surprisingly, software houses and IT departments, which had already adopted agile methods, coped quickly with the pandemic situation by virtualizing agile practices, digitizing agile artifacts, and sending their employees to work from home [3–8]. Nevertheless, as team collocation is one of

the pillars of agile software development [9], an ad-hoc transition to a remote environment challenged the well-established approach to delivering product increments. Indeed, many of the collaborative practices that used to depend on face-to-face communication were rapidly disrupted [10, 11] in the new reality, even though remote communication itself was to some extent commonly practiced before the pandemic [8, 12]. As a consequence, a short-term drop in performance occurred immediately after the transition to a remote environment [3, 13].

The phenomenon of the initial drop in performance is evidenced by the results of three surveys conducted during the first months of the COVID-19 pandemic. In April 2020, Ralph et al. surveyed software developers who switched from working in an office to working from home because of COVID-19 [14]. Based on 2225 responses that met the inclusion criteria, the authors concluded that perceived productivity declined. Another survey study conducted in Germany almost during the same time period and mainly focused on managers and project management experts (with a total of 171 responses) found a small perceived loss in productivity after switching to remote work [15]. The decreased productivity due to the pandemic was also reported by Butt et al., who surveyed over 250 software developers, team leaders, and project managers between April and June 2020 [1].

As agile methods provide no guidelines for remote work, agile teams needed time to come up with in-house solutions. Indeed, recent studies show that the performance of agile teams has not permanently decreased [2, 4, 13]. Nonetheless, despite a significant body of literature that focuses on switching to remote work and the resulting impacts on agile teams (for review, see [6, 7]), only a few studies have examined how agile teams may overcome the new challenges [14, 16, 17]. Furthermore, the mitigation strategies and adjustments are quite diverse, suggesting that they depend on many factors and the individual situation of the team (e.g., effects due to the maturity of the agile process in use) [6]. Therefore, the agile community is responsible for elaborating and reporting context-specific strategies and best practices for remote agile teams. In light of this need, we report on an Action Research project carried out in Lufthansa Systems Poland (1) to revisit their ad-hoc adaptations to remote work; and (2) to elaborate systematic long-lasting solutions for maintaining efficiency in such settings. To guide our work, we raised the following research questions:

RQ1: How did Scrum teams adapt their practices and processes due to the ad-hoc shift to remote work?

RQ2: What are the advantages of remote work for Scrum team members?

RQ3: What new challenges are faced by virtual Scrum teams and their members?

RQ4: How can these challenges be mitigated?

The main contribution of our research is twofold: (1) developing mitigation strategies for Scrum teams to tackle challenges posed by remote working; and (2) enhancing knowledge regarding agile software development in the post-COVID-19 era.

## 2 Method and Setting

The study followed the canonical action research (AR) within the field of software engineering [18, 19]. It stretched over a six-month period, during which two AR cycles were run. Each AR cycle had the following phases: Diagnosing, Action Planning, Action Taking, Evaluating, and Specifying Learning. Table 1 shows how data was collected in the different phases of AR cycles. The research process itself was hosted at Lufthansa Systems Poland – a subsidiary founded in 1998 in Gdansk, Poland. Two teams namely *Covid-DI* and *Group App* participated in the study. Their composition is summarized in Table 2. Although both teams belonged to the same branch (i.e., the Digital Delivery Lab), where solutions supporting the Lufthansa Group’s digital infrastructure were developed, they did not collaborate with each other. However, they used the same tools to create test flight numbers and communicated with the team responsible for cloud infrastructure and the Continuous Integration and Delivery (CI/CD) process. They also used Scrum both before and after the transition to remote working.

**Table 1.** Data collection techniques.

AR phase	Objective	Technique	Data source
<i>Diagnosing</i>	Preliminary identification of existing issues	Non-structured interview	Team members
	Confirming the issues identified	Online survey	
<i>Action Taking</i>	Inspecting implemented interventions	Participatory observation	Researcher
<i>Evaluating</i>	Assessing the interventions	Online survey	Team members
	Discussing intervention performance and collecting recommendations	Focus group	

The *Covid-DI* team has developed a system that provides automation of the process of verifying documents that are required for a flight. The system features microservices and an internal web application. It receives a passenger’s data along with all the documents provided by the passenger through a web application<sup>1</sup> developed by another team. After entering personal data and ticket number, the passenger is presented with a list of possible combinations of documents to be sent to receive a positive verification.

The *GroupApp* team rolls out a mobile app for SWISS, Austrian, and Brussels Airlines (a single application with three visual overlays in accordance with the expectations of the individual carriers belonging to the Lufthansa Group<sup>2</sup>). The application stores flight information and enables an end user to review one’s trip details. It is a personal

<sup>1</sup> Available at: <https://www.lufthansa.com/ge/en/online-check-in>.

<sup>2</sup> A version customized for Austrian Airlines is available at: <https://www.austrian.com/us/en/austrian-app>.

assistant that offers a real-time display of relevant travel information and keeps travelers up to date with flight notifications. The app's functionality covers mobile check-in, changing or reserving a seat, and handling boarding passes.

**Table 2.** Professionals taking part in the study.

Covid-DI	GroupApp
1 × Team Leader	1 × Team Leader
1 × Product Owner	1 × Product Owner
1 × Scrum Master	1 × Scrum Master
5 × Backend Developer	4 × Backend Developer
1 × Frontend Developer	1 × Frontend Developer
1 × Junior Test Engineer	1 × Test Engineer
1 × Business Analyst	1 × Business Analyst
1 × System Architect	1 × System Architect
	1 × UX/UI Designer

## 3 Findings

### 3.1 The First Action Research Cycle

Per *Diagnosing*, we determined that Microsoft Teams had been established as the primary videoconferencing platform after the transition to remote work. Usually, the Scrum Master shared his screen presenting the Scrum board with the project backlog, current user story, or tasks to be discussed at Daily Meetings or during Backlog Refinement. Retrospectives were conducted using Timbo and TeamRetro tools for the *Covid-DI* and the *Group App* teams, respectively. Sprint planning was achieved through the use of screen sharing, and the PlanITPoker tool, which supports collective estimation with Planning Poker [20, 21], was used for task estimation. Given its visualization-oriented capabilities, both teams also introduced the online Miro whiteboard to facilitate various types of meetings, brainstorming, discussions, etc.

As the *Covid-DI* team is concerned, a few significant changes were reported by the Scrum Master compared to Scrum practices implemented in the on-site office environment. Daily Scrum meetings became more static. Everyone presented their progress without much thought to any discussion or exchange of information (which was an indispensable part when hosting those in the office). Therefore, at the end of each meeting, there was a moment to discuss ongoing problems, if any. Although, probably due to the lack of visible feedback from the other team members, there was usually silence – which was rare at office meetings. Another significant change was the introduction of two additional meetings. A non-mandatory Open Session meeting was to be held 3 times a week (Monday, Wednesday, and Friday) for the development team and collaborating teams or individuals. The meeting was created due to the need to pass information between teams. The second meeting set up in the remote environment was Operation Weekly – a

meeting between developers and individual DevOps team members to discuss resource and infrastructure issues. The regularity of Sprint Review meetings also deteriorated relative to the on-site environment. In the remote mode, the meeting used to take place every two to three sprints, and over time this regularity has diminished to an occasional occurrence. In contrast, at the office Sprint Review was held every sprint, and it was seldom canceled – only if a particular sprint did not produce any presentable Increment. In the *GroupApp* project, the only change was to hold Sprint Review every second iteration to save the development team’s time.

A survey fueled by prior non-structured interviews revealed both positive and negative feedback regarding Scrum performance in a remote environment (Table 3). The main conclusions are as follows: (1) working from home hinders both communication between teams and ad-hoc communication between team members; (2) teams’ commitment is lower in a remote environment; (3) it becomes increasingly difficult to integrate a new employee into a virtual team; and (4) being physically together in the open space office favors the development of innovative solutions.

**Table 3.** Selected qualitative feedback – the first AR cycle.

Setting fit	<p>— <i>“Remote: to pursue tasks. On-site: for creative work”</i> [System Architect]</p> <p>— <i>“Working on-site is better when a project requires a lot of creativity and innovation. Such a setting helps create more team involvement which is necessary when there are a lot of unknowns. For more delivery-oriented projects or when there are fewer unknowns and it’s easy to plan the work, the remote setting is convenient as the tools currently available make managing work easy”</i> [Developer]</p> <p>— <i>“On-site is better for coming up with something out-of-the-book; remote is better for earning more from the project”</i> [Developer]</p> <p>— <i>“Both methods of conducting Scrum have their advantages and disadvantages. A lot depends on the project, its complexity, and the people involved. When a project is developed by people from multiple locations, remote work gains more advantages. In the case of a project developed by a team in a single location, physical collocation can have a number of advantages. Of course, we must also take into account individual characteristics of team members: some simply work better in the office, others remotely”</i> [Developer]</p>
Communication and maintaining focus	<p>— <i>“When working remotely during meetings, I am able to prepare lunch or do some exercises – albeit it is much harder for interpersonal contact”</i> [Developer]</p> <p>— <i>“Certainly, the role of the Scrum master in terms of the ‘soft’ part is made more difficult, because people – as a rule – prefer not to turn the cameras on. Muting the microphone makes team members less willing to speak up; and even often by the time they manage to turn on the microphone, the meeting has already moved on to another topic. Without seeing the body language of others, it is difficult to determine what a given member would prefer, how they feel, and what they actually think about a topic. Meetings cease to be relaxed and become more focused on the work and the task. This is nice at first, but in the long run, it erodes morale, and with that also the quality of work/creativity. Bringing in new members is especially challenging. Established staff may not see that much of a difference, while it is extraordinarily difficult for newcomers to get acquainted with a project and feel like they are a part of the team”</i> [Scrum Master]</p>

(continued)

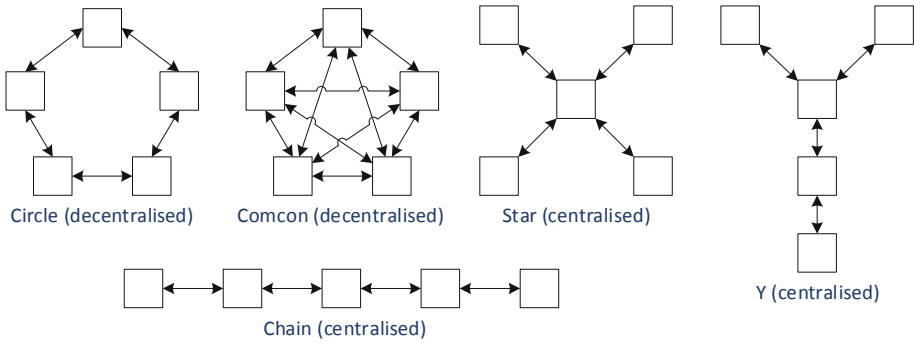


**Table 3.** (continued)

Communication and maintaining focus (cont.)	<p>— “I see such a benefit in remote work that I can attend a meeting and do something else in parallel. Normally that would not be welcome. Remotely, what the eyes do not see... Being co-located in the same office facilitates communication outside of the process. You turn around in your chair, shout that something is not working, and a discussion is swiftly established” [Developer]</p> <p>— “Working in the office I have always spent half a day at meetings, and the time was often wasted. Now, as the meeting moderately concerns me, I can do a task, at the same time knowing what is being said there” [Developer]</p> <p>— “On-site office environment: you want to do something, but someone asks if you are going out for coffee. So the job rests. But during that coffee, there may be an ‘unplanned’ exchange of ideas and something interesting comes up. It is a toss-up. If I were developing any sort of app of my own, I would definitely prefer the core team to have direct contact. I myself somewhat nostalgically remember those ad-hoc meetings, when you would gather with the team in a small cubicle: people armed only with their brains and crayons to draw on the board – and we would work out an issue. The best part of the job, as far as I am concerned, is gone now. I also get the impression that we used to try to analyze an issue in-depth, consider various possibilities, and finally choose the optimal solution. And now there is a ‘put it into production ASAP’ policy in place” [System Architect]</p> <p>— “On-site office environment: the team gets together and comes up with a solution. Everyone then knows why it was done that way. Remotely: a person comes up with a solution and presents it ostensibly for review. And the reviewer is actually busy with something else (he/she has one’s tasks on the board) and most often just pats it down because it is also inappropriate to say ‘do it completely differently’. So maybe it is that the transition to remote has simply intensified this” [Developer]</p>
Perceived efficiency	<p>— “I have a somewhat similar feeling that it is easier to stick to the schedule and efficiency of meetings when Scrumming remotely” [Developer]</p> <p>— “I have said again and again that there are too many meetings and that people are invited who should not/do not want to be there. In fact, it is not an on-site/remote Scrum problem, but if you cannot change this (and you simply cannot), it is easier to cope with remotely (for me). In addition, all folks are by the computer: I can write to someone and quickly find out from a given person about what and how. Also, it depends on the person/project /team. I would gain nothing being on-site except the stress of being stuck in traffic” [Developer]</p> <p>— “It seems to me that working on-site makes less sense than working from home. My main point is that better organization of work in remote mode is absolutely necessary, so introducing such a mode significantly improves work, communication, and team engagement. What I have not quite witnessed at the office, where it was more perceived (at least from my observation) as redundancy of meetings... Because we communicate all the time anyway” [Business Analyst]</p> <p>— “I did not have a chance to work in an office environment, but in general I am sure I am more efficient remotely (the fact is that I have decent working conditions at home). In the office, there are a lot of distractions and I am just able to get more done working remotely” [Developer]</p>

The results of the *Diagnosing* phase prompted the research team to prioritize the issue of improving team communication patterns within *Action Planning*. To this end, communication patterns and practices used by the teams at that time were scrutinized. In both cases, communication patterns used were akin to the decentralized Comcon model

(Fig. 1). In the Comcon everyone is free to communicate with the rest of the team [22]. Besides, cross-team information exchange was not governed by any rules; developers contacted professionals from other teams asynchronously. This often led to multiple repetitions of a given piece of information within the surveyed teams and sometimes resulted in repeated queries to people from other teams.

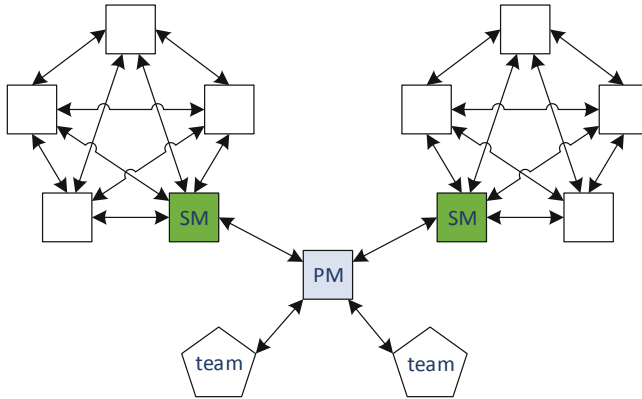


**Fig. 1.** Team communication models [22].

To address the problem, teams were presented with three centralized models – Star, Y, and Chain (Fig. 1). Upon discussion within the teams, it was agreed that internal communication should remain free-form and become as direct as possible, as typical for the Comcon model. However, cross-team communication should be handled by designated individuals. A developer with the deepest knowledge in every team assumed the role of Service Manager (SM) and was assigned the responsibility of aggregating information from other teams and answering immediately teammates' questions that did not require third-party involvement. Henceforth, other developers were instructed to pass on cross-team inquiries to the SM and avoid contacting collaborating teams on their own. In order to improve decision-making at the cross-team level, it was also decided that SMs would not forward inquiries directly to their counterparts from other teams, but to the Project Manager instead – who was also able to answer some questions at once. If necessary, it was the latter's responsibility to gather information from other teams. Thus, cross-team communication became aligned with the centralized Star model and the overall communication model took on a hybrid form (Fig. 2).

In order to make the process of transferring information between teams more transparent, documentation was proposed, describing who, on which team, is responsible for which specific area, and to whom (if necessary) to direct specific questions. On top of that, each team under study along with collaborating teams was advised to create an extensive Q&A to keep question repetitions in check. To ensure a regular exchange of information between teams, periodic meetings were stipulated for Service and Project Managers. The meetings were designed as optional since there was not always a need to pass information. That said, the calendar placeholder itself ensured that everyone was available at the same time. Finally, separate channels on the MS Teams platform (dedicated to specific areas of the project) were set up. One of those was to be used to communicate internal information regarding merge requests and tasks awaiting review.

The other was dedicated to solving technical problems from a programming standpoint. Unlike the former channel which did not have a clearly specified post structure, the latter came with a template with several fields necessary for the inquirer to complete. The fields included the JIRA task number, component, technology, and possible working solutions.

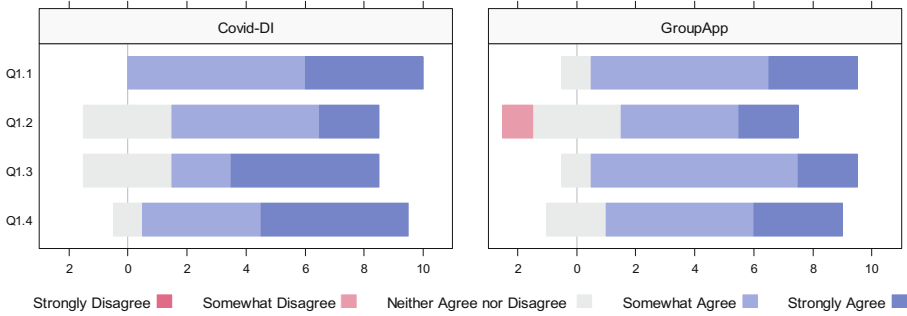


**Fig. 2.** Hybrid team communication solution.

Regarding the difficulties with integrating new employees, it was proposed to work co-located in the office for one day a week. Such on-site workdays would allow for organizing most of the Scrum ceremonies as face-to-face meetings and taking advantage of effective communication opportunities. Presenting stories to jointly work out solutions, discussing the advantages and disadvantages of a given variant, fostering familiarity between employees, and creating a positive team vibe contribute to the quality of the end product. Teams were not forced to have a predetermined day of the week on which they should come to the office. The decision on the chosen day was still in the hands of the team and resulted from needs at a given time – which allowed the team to maintain its autonomy. Lastly, although *Diagnosing* clearly pointed towards a decrease in team engagement, due to the already broad scope of changes being planned to introduce, it was decided to address this issue in the next AR cycle.

*Action Taking* spanned across three Sprints. After a total of six weeks, the interventions were assessed. *Evaluating* gathered feedback on the following aspects (Fig. 3):

- Q1.1 – Appointing Service Managers to relay information improved communication between teams;
- Q1.2 – Setting up optional meetings between Service Managers and the Project Manager improved knowledge sharing between teams;
- Q1.3 – Establishing a new, explicit communication model that specifies with whom, in what case to communicate improved communication efficiency;
- Q1.4 – Introducing additional channels on the MS Teams platform dedicated to specific areas of the project improved ad-hoc communication.



**Fig. 3.** Aggregated results of the evaluation survey conducted during the first AR cycle.

Eventually, given both the performance of both teams across the Sprints and the feedback, *Specifying Learning* led to the following conclusions:

1. **Establishing an explicit communication model and appointing Service Managers to relay information facilitate cross-team communication.** The remote environment has almost deprived ad-hoc communication, which was done naturally in the open space office environment (the proverbial “spin on the chair”). By implementing clearly defined communication patterns and identifying the appropriate individuals responsible for gathering and relaying external information, the teams were able to communicate more efficiently. The new approach not only reduced the number of questions that went unanswered for prolonged periods of time but also decreased the number of developers that one had to disturb before reaching the developer who actually knew the answer.
2. **Joint meetings between Service Managers and the Project Manager improve knowledge sharing.** Providing a convenient time for all Service Managers to talk to each other affects the overall awareness of the activities of the collaborating teams and allows immediate decisions to be made on topics affecting their work. In addition, it ensures that key stakeholders will have time to deal with issues affecting other teams, which are very often set aside. A faster and more transparent flow of information also ensures a reduction in possible errors resulting from misunderstandings between teams.
3. **Introducing additional areas for specific types of information improves communication.** One of the problems encountered in the remote environment was information preservation and duplication. Establishing additional channels on the MS Teams platform improved the response rate of teammates on specific issues and ensured that the knowledge transferred was, to some extent, preserved.

### 3.2 The Second Action Research Cycle

Upon completing the intervention, it was decided that the following two-week sprint should constitute a hiatus between the research cycles. This allowed both teams to unwind from additional workflows (attending focus groups, completing surveys), and to consolidate previously implemented solutions before the next cycle.

Although the discussion regarding Scrum's deficiencies in the remote environment and the extent to which the main issues have been addressed by the already implemented measures proved to be vigorous, members of both teams mostly agreed that the quality of the meetings suffered in the remote environment, while the transparency of the teams' work deteriorated. They supported this *Diagnosis* primarily by pointing out frequent occurrences of silence during meetings ("as if no one knows who is supposed to speak up now"), reduced interactions, and greater variability in the online meeting toolset used (e.g., various applications for conducting retrospectives). The *Covid-DI* team considered Daily Scrum to be the most problematic meeting, whereas the *Group App* team highlighted Sprint Retrospective. However, both teams specifically referenced both of these ceremonies. In addition to the previously mentioned problems, participants also hinted that Sprint Retrospective was getting overlong for them, and they were losing interest in it. Several people felt that "retros were held for the sake of holding them". The aforementioned factors primarily caused a big decline in the amount of information exchange.

Moreover, the majority of participants agreed with the statement that regardless of the ceremony, in the remote environment, they are more likely to miss significant information and have to spend more time inquiring about the issues of specific people even just after the meeting is over. Those with more seniority in the industry and experience working in Scrum noted that the principles of the framework in a remote environment began to deteriorate. They highlighted the increased number of meetings resulting from the need for re-discussions, the failure to keep meetings within time constraints, the resulting necessity to catch up on any suspended meetings, and conducting some ceremonies (e.g. Sprint Retrospective) not in line with the Scrum Guide.

As further scrutiny confirmed an increase in the volume of information provided while decreasing the number of tangible details, one of the measures proposed within *Action Planning* was to keep webcams active during meetings. By observing someone's gestures or facial expressions, others can effectively assess whether a message has been understood and whether the audience agrees with the statement made. On top of that, to enhance the flow of conversations during meetings, individuals working in a quiet environment were expected to keep their microphones on to avoid wasting time unmuting themselves. Oftentimes switching to the application window and hitting the unmute button proved to take up enough time for the speaker to move on to the next topic, resulting in disregarding any concerns from the team. Additionally, it was agreed to allocate the last three minutes of a meeting to a brief summary. At the Daily Scrum, Scrum Master was to recapitulate crucial information that had been given during the meeting and to make sure that everyone had grasped it. During the Sprint Retrospective, both the completed action points and newly established ones were to be recapped at the end.

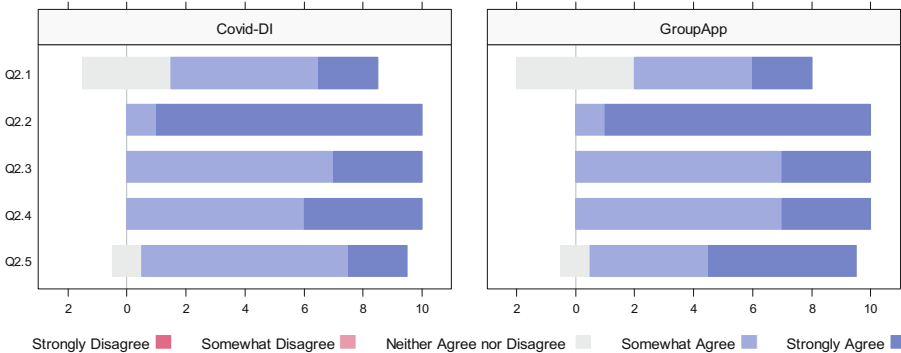
As for maintaining the principles of Scrum, i.e. transparency and self-organization, it was decided to conduct Q&A-heavy single-day training sessions for both teams on the Scrum framework. The training sessions, which were conducted by the Scrum Masters, followed a uniform agenda consisting of the following: (1) an overview of various project management methods; (2) the Agile Manifesto; (3) the Scrum framework, including its

principles, artifacts, ceremonies, roles, and strengths; (4) Scrum vs. SAFe; and (5) a summary.

Notwithstanding the above, in order to address the issue of the lack of engagement, which was identified in the first cycle but left unresolved, it was decided to introduce workshops to better understand the project and the needs of the teams. The workshops were held on-site, as an opportunity presented itself for the management to get to know people working for a shorter time than the rest. During this time, meetings were held with team leaders to convey the broad vision of the project. Professionals were given an opportunity to express in what direction they would like to develop, what they like about the project, and what is missing. Furthermore, even though it was not evaluated in the first cycle due to a too-short time horizon, the rule of one working day in the office to facilitate the onboarding process of newcomers and the socialization of team members was maintained.

After another three Sprints of the *Action Taking* phase, the *Evaluating* phase took place to determine whether the following actions had the desired effects (Fig. 4):

- Q2.1 – Running workshops to discuss the project and the needs of the team increased team engagement;
- Q2.2 – Having the team co-located in the office once a week accelerated the integration of newcomers;
- Q2.3 – Creating a meeting summary facilitated retention of key aspects;
- Q2.4 – turning on cameras and microphones during meetings improved signaling of concerns and understanding of the information provided;
- Q2.5 – Conducting Scrum training improved adherence to the established principles of the framework.



**Fig. 4.** Aggregated results of the evaluation survey conducted during the second AR cycle.

The feedback was overwhelmingly positive. Thus, *Specifying Learning* phase completed the intervention with the following lessons learned:

1. **The way online meetings are conducted affects the effectiveness of communication.** Keeping webcams switched on allows for more natural and nuanced communication by enabling team members to observe each other's reactions and body

language. This can enhance engagement, build trust, and foster a smoother exchange of information. In addition, the introduction of a brief summary at the end of meetings facilitates better recollection, which, in turn, leads to greater awareness among team members of the issues at hand.

2. **A better understanding of Scrum enhances the team's ability to collaborate effectively.** A remote environment exposes dysfunctions within a team. Fixing the Scrum process requires a comprehensive knowledge of the framework, which can be obtained by participating in training sessions.
3. **Workshops that discuss the project vision and the team's needs have a positive impact on team commitment.** Such workshops provide a shared understanding of the project and its context, foster team bonding, and encourage openness among individuals.
4. **Requiring workers to come to the office one day per week constitutes a healthy compromise that balances work flexibility, employee integration, and collaboration quality through in-person sessions.** This practice not only enables most Scrum ceremonies to be organized on-site but also fosters spontaneous conversations. During such conversations, employees occasionally tend to move away from work-related topics, which promotes the growth of social connections.
5. **Not all members of an agile team necessarily have an agile mindset.** Typically, some developers are hesitant to exchange information or share knowledge, and this attitude may persist regardless of the working environment. In fact, our prior research [23] in an on-site environment revealed the same findings. However, in a remote setting, this reluctance can further exacerbate communication and collaboration challenges with these individuals.

## 4 Discussion

### 4.1 How Did Scrum Teams Adapt Their Practices and Processes Due to the Ad-Hoc Shift to Remote Work?

Numerous studies have reported that to transition to remote work, agile teams implemented virtualization of work using software tools [2, 5, 6, 8, 15]. Video conferencing platforms such as Microsoft Teams and Zoom have enabled Scrum meetings to be conducted remotely, while online whiteboard-based collaboration tools like Miro and Mural have facilitated collaboration. It is not surprising that the ad-hoc shift to remote work in the participating teams looked quite similar. Additionally, our teams have introduced dedicated tools that incorporate collaborative games (proven successful in face-to-face meetings [9, 23–25]) for effective Sprint Retrospective and Sprint Planning. Interestingly, this approach stood out, as Neumann & Bogdanov [6] found in their SLR that other teams relied solely on the chat functionality within video conferencing tools and, consequently, lost the playful nature of their meetings. Additionally, one of our teams implemented new types of meetings to coordinate work with collaborating teams.

### 4.2 What are the Advantages of Remote Work for Scrum Team Members?

Several advantages of remote work were highlighted by the participants of our study, which can be categorized into two main groups: increased flexibility and improved productivity. In fact, the latter may be an outcome of the former, as greater flexibility results

in happier developers, and empirical evidence from previous studies has shown that happy developers are more productive [26]. With a work-from-home policy, software engineers enjoy greater control over their work schedule, resulting in higher job satisfaction and a better work-life balance. Accordingly, the majority of them would like to continue working remotely, which indicates an improvement in their well-being. Moreover, remote work enables them to be more productive by reducing distractions and interruptions while enhancing their ability to focus on the tasks at hand. Furthermore, remote work saves time that would otherwise be spent on commuting. These findings are in line with the results of several previous studies [3–5, 15, 27], some of which even suggest that employees can no longer imagine switching to pure co-located work [15]. Nonetheless, some authors have emphasized the negative aspects associated with remote work. Ralph et al. found that the pandemic had a negative effect on developers' well-being and productivity [14], while Butt et al. reported that the investigated team members experienced increased mental and physical stress [1]. Additionally, Griffin [17] described the challenge of household distractions for agile team members working from home. Finally, it was suggested that the work-life balance was disrupted by even more blurred boundaries between work and life [7].

### **4.3 What New Challenges are Faced by Virtual Scrum Teams and Their Members?**

Despite the availability of various tools to support both synchronous and asynchronous communication, the absence of face-to-face interaction introduced a lot of inefficiency to the exchange of information and required the investigated teams to put extra effort into collaboration. Additionally, many participants in our study noticed that remote work hindered creativity since team members became less interconnected. Furthermore, without regular in-person interactions, both teams experienced a significant decrease in social exchange, which resulted in a reduced sense of team cohesion. These observations are consistent with the results of several previous studies [5–7, 16, 28]. Another challenge that arose in a remote environment was the onboarding process of new team members. This challenge encompassed both the need to assimilate newcomers into the team dynamic, as well as to ensure their understanding of established processes, practices, and tools. Prior studies [3, 5, 6, 29] also observed this challenge. Finally, in both participating teams the transparency of the team's work deteriorated, requiring special actions to restore it. This finding contradicts previous studies [2, 3, 5], which reported that the overall transparency increased as a result of the digitization of artifacts and the increased use of digital chat channels.

## **5 Conclusions**

The COVID-19 pandemic revealed that many preconceptions about remote work were misplaced. What is more, even though the pandemic is behind us, there is a global desire to retain the flexibility of remote work. Our research confirms the numerous benefits of remote work that have been identified in previous studies. Working from home allows for increased employee comfort and greater personalization of one's work environment,



creating opportunities to better align personal and professional needs. Moreover, working remotely eliminates the time and expense associated with daily commuting to a physical office. Nevertheless, the ad-hoc switch to remote work also presented challenges for agile teams, such as a lack of proper communication and reduced team cohesion. In this work, we report on an Action Research project in Lufthansa Systems Poland, where we worked with two Scrum teams to systematically address challenges posed by the remote environment. Our collaboration resulted in the following solutions:

- establishing an explicit communication model between collaborating teams;
- setting up optional meetings for representatives of collaborating teams and the Project Manager;
- introducing one on-site workday per week;
- summarizing meetings at the end of the session;
- keeping webcams on during online meetings;
- establishing additional channels on the MS Teams platform to maintain knowledge;
- organizing occasional workshops to discuss the project vision and the team's needs.

Our research not only demonstrates once again that Scrum is agile itself and thus can be applied outside of its traditional boundaries (previously Scrum has its proved flexibility in adapting to large-scale projects [30–33]) but also shows that Scrum in an online environment does not lose its benefits.

The study has two main limitations. Firstly, the evaluation of the implemented solutions was done subjectively. Secondly, there is a potential bias among study participants. Employees who already hold the belief of working exclusively remotely may not have viewed on-site meetings positively and could have disregarded their positive aspects.

## References

1. Butt, S.A., Misra, S., Anjum, M.W., Hassan, S.A.: Agile project development issues during COVID-19. In: Przybyłek, A., Miler, J., Poth, A., Riel, A. (eds.) LASD 2021. LNBIP, vol. 408, pp. 59–70. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-67084-9\\_4](https://doi.org/10.1007/978-3-030-67084-9_4)
2. Marek, K., Wińska, E., Dąbrowski, W.: The state of agile software development teams during the Covid-19 pandemic. In: Przybyłek, A., Miler, J., Poth, A., Riel, A. (eds.) LASD 2021. LNBIP, vol. 408, pp. 24–39. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-67084-9\\_2](https://doi.org/10.1007/978-3-030-67084-9_2)
3. Neumann, M., Bogdanov, Y., Lier, M., Baumann, L.: The Sars-Cov-2 pandemic and agile methodologies in software development: a multiple case study in germany. In: Przybyłek, A., Miler, J., Poth, A., Riel, A. (eds.) LASD 2021. LNBIP, vol. 408, pp. 40–58. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-67084-9\\_3](https://doi.org/10.1007/978-3-030-67084-9_3)
4. Kettunen, P., Gustavsson, T., Laanti, M., Tjernsten, A., Mikkonen, T., Männistö, T.: Impacts of COVID-19 Pandemic for software development in nordic companies – agility helps to respond. In: Gregory, P., Kruchten, P. (eds.) XP 2021. LNBIP, vol. 426, pp. 33–41. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-88583-0\\_4](https://doi.org/10.1007/978-3-030-88583-0_4)
5. Neumann, M., Bogdanov, Y., Sager, S.: The Covid 19 pandemic and its effects on agile software development. In: 5th International Conference on Software Engineering and Information Management (ICSIM), pp. 51–60. ACM, New York, NY (2022). <https://doi.org/10.1145/3520084.3520093>

6. Neumann, M., Bogdanov, Y.: The impact of Covid-19 on agile software development: a systematic literature review. In: 55th Hawaii International Conference on System Sciences, pp. 7350–7359. University of Hawai‘i, Mānoa, HI (2022). <https://doi.org/10.24251/HICSS.2022.882>
7. Ozkan, N., Erdil, O., Gök, M.Ş: Agile teams working from home during the Covid-19 pandemic: a literature review on new advantages and challenges. In: Przybyłek, A., Jarzębowski, A., Luković, I., Ng, Y.Y. (eds.) LASD 2022. LNBIP, vol. 438, pp. 38–60. Springer, Cham (2022). [https://doi.org/10.1007/978-3-030-94238-0\\_3](https://doi.org/10.1007/978-3-030-94238-0_3)
8. Wang, X., Hubner, S., Melegati, J. et al.: Startup Digi-Dojo: a digital space supporting practice and research of startup remote work. In: International Conference on Software Business, Bolzano, Italy (2022)
9. Ng, Y.Y., Skrodzki, J., Wawryk, M.: Playing the sprint retrospective: a replication study. In: Przybyłek, A., Morales-Trujillo, M.E. (eds.) LASD/MIDI -2019. LNBIP, vol. 376, pp. 133–141. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-37534-8\\_7](https://doi.org/10.1007/978-3-030-37534-8_7)
10. Smite, D., Mikalsen, M., Moe, N.B., Stray, V., Klotins, E.: From collaboration to solitude and back: remote pair programming during COVID-19. In: Gregory, P., Lassenius, C., Wang, X., Kruchten, P. (eds.) XP 2021. LNBIP, vol. 419, pp. 3–18. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-78098-2\\_1](https://doi.org/10.1007/978-3-030-78098-2_1)
11. Matthies, C., Teusner R., Perscheid M.: Challenges (and Opportunities!) of a Remote Agile Software Engineering Project Courseduring COVID-19. In: 55th Hawaii International Conference on System Sciences, pp. 1–10. University of Hawai‘i, Mānoa, HI (2022). <https://doi.org/10.24251/HICSS.2022.113>
12. Jarzębowski, A., Sitko, N.: Communication and documentation practices in agile requirements engineering: a survey in polish software industry. In: Wrycza, S., Maślankowski, J. (eds.) SIGSAND/PLAIS 2019. LNBIP, vol. 359, pp. 147–158. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-29608-7\\_12](https://doi.org/10.1007/978-3-030-29608-7_12)
13. Connor, M., Conboy, K., Dennehy, D.: COVID-19 Affected remote workers: a temporal analysis of information system development during the pandemic. *J. Decis. Syst.* **31**(3), 207–233 (2022). <https://doi.org/10.1080/12460125.2020.1861772>
14. Ralph, P., et al.: Pandemic programming. *Empir. Softw. Eng.* **25**(6), 4927–4961 (2020). <https://doi.org/10.1007/s10664-020-09875-y>
15. Schmidtnr, M., Doering, C., Timinger, H.: Agile working during COVID-19 pandemic. *IEEE Eng. Manag. Rev.* **49**(2), 18–32 (2021). <https://doi.org/10.1109/EMR.2021.3069940>
16. Da Camara, R., Marinho, M., Sampaio, S., Cadete, S.: How do agile software startups deal with uncertainties by Covid-19 pandemic? *Int. J. Softw. Eng. its Appl.* **11**(4), 15–34 (2020). <https://doi.org/10.5121/ijsea.2020.11402>
17. Griffin, L.: Implementing lean principles in scrum to adapt to remote work in a Covid-19 impacted software team. In: Przybyłek, A., Miler, J., Poth, A., Riel, A. (eds.) LASD 2021. LNBIP, vol. 408, pp. 177–184. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-67084-9\\_11](https://doi.org/10.1007/978-3-030-67084-9_11)
18. Staron, M.: Action Research in Software Engineering. Springer International Publishing (2020). <https://doi.org/10.1007/978-3-030-32610-4>
19. Marcinkowski, B., Gawin, B.: A study on the adaptive approach to technology-driven enhancement of multi-scenario business processes. *Inf. Technol. People* **32**(1), 118–146 (2019). <https://doi.org/10.1108/ITP-03-2018-0142>
20. Butt, S.A., Ercan, T., Binsawad, M., et al.: Prediction based cost estimation technique in agile development. *Adv. Eng. Softw.* **175**, 103329 (2023). <https://doi.org/10.1016/j.advengsoft.2022.103329>
21. Butt, S.A., Khalid, A., Ercan, T., et al.: A software-based cost estimation technique in scrum using a developer’s expertise. *Adv. Eng. Softw.* **171**, 103159 (2022). <https://doi.org/10.1016/j.advengsoft.2022.103159>

22. Pennington, D.C.: *The Social Psychology of Behavior in Small Groups*, 1st edn.. Routledge (2002). <https://doi.org/10.4324/9781315787800>
23. Przybyłek, A., Albecka, M., Springer, O., Kowalski, W.: Game-based Sprint retrospectives: multiple action research. *Empir. Softw. Eng.* **27**(1), 1–56 (2021). <https://doi.org/10.1007/s10664-021-10043-z>
24. Wawryk, M., Ng, Y.Y.: Playing the Sprint Retrospective. In: 14th Federated Conference on Computer Science and Information Systems (FedCSIS'19), Leipzig, Germany (2019). <https://doi.org/10.15439/2019F284>
25. Mich, D., Ng, Y.Y.: Retrospective games in intel technology Poland. In: 15th Federated Conference on Computer Science and Information Systems (FedCSIS'20), Sofia, Bulgaria (2020). <https://doi.org/10.15439/2020F62>
26. Graziotin, D., Fagerholm, F., Wang, X., Abrahamsson, P.: What happens when software developers are (Un) happy. *J. Syst. Softw.* **140**, 32–47 (2018). <https://doi.org/10.1016/j.jss.2018.02.041>
27. Russo, D., Hanel, P.H.P., Altnickel, S., van Berkel, N.: Predictors of well-being and productivity among software professionals during the COVID-19 pandemic – a longitudinal study. *Empir. Softw. Eng.* **26**(4), 1–63 (2021). <https://doi.org/10.1007/s10664-021-09945-9>
28. Deshpande, A., Sharp, H., Barroca, L., Gregory, P.: Remote working and collaboration in agile teams. In: International Conference on Information Systems, Dublin, Ireland (2016)
29. Nolan, A., et al.: To work from home (WFH) or not to work from home? lessons learned by software engineers during the COVID-19 pandemic. In: Yilmaz, M., Clarke, P., Messnarz, R., Reiner, M. (eds.) EuroSPI 2021. CCIS, vol. 1442, pp. 14–33. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-85521-5\\_2](https://doi.org/10.1007/978-3-030-85521-5_2)
30. Kowalczyk, M., Marcinkowski, B., Przybyłek, A.: Scaled agile framework. Dealing with software process-related challenges of a financial group with the action research approach. *J. Softw. Evol. Process*, **34**(6), e2455 (2022). <https://doi.org/10.1002/smr.2455>
31. Kalenda, M., Hyna, P., Rossi, B.: Scaling agile in large organizations: practices, challenges, and success factors. *J. Softw. Evol. Process* **30**(10), e1954 (2018). <https://doi.org/10.1002/smr.19541>
32. Buchalceva, A., Dolezel, M.: Examining the Usage of Scaled Agile Methods in the Czech Republic. In: 29th International Conference on Information Systems Development (ISD2021), Valencia, Spain (2021)
33. Joskowski, A., Przybyłek, A., Marcinkowski, B.: Scaling scrum with a customized nexus framework: a report from a joint industry-academia research project. *Softw.-Pract. Exp.* (2023). <https://doi.org/10.1002/spe.3201>

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# Waste Self-reporting for Software Development Productivity Improvement

Marc Sallin<sup>1</sup>(✉), Martin Kropp<sup>1</sup>, Craig Anslow<sup>2</sup>, and Robert Biddle<sup>3</sup>

<sup>1</sup> University of Applied Sciences Northwestern Switzerland, Windisch, Switzerland  
marc.sallin@outlook.com, martin.kropp@fhnw.ch

<sup>2</sup> Victoria University of Wellington, Wellington, New Zealand  
craig.anslow@vuw.ac.nz

<sup>3</sup> Carleton University, Ottawa, Canada  
robert.biddle@carleton.ca

**Abstract.** Little research has been done on enabling software development teams to self-report waste to assist in productivity improvement. This study created a waste categorization and survey for teams to identify and quantify wasteful activities. Developers from a Swiss company used the survey for three weeks. Participants found the survey helpful for identifying waste but there was little evidence that self-reported waste correlated with improved performance.

**Keywords:** Empirical · Software development · Productivity · Efficiency · Waste · Lean · Case Study

## 1 Introduction

Improving the productivity of software development teams is a natural and perpetual goal for organizations. However, measuring the productivity of software development teams has always been seen as inherently difficult. Productivity is determined as the ratio of an achieved output and the required input [1]. In software development, the input mainly consists of the working time spent on development, expressed as the cost of developing software. With the output, it is less clear: the produced lines-of-code or number of implemented functions have shown not to be suited to measure the output. Both quantity and quality of the output are hard to define for software development and typically vary from project to project. Therefore, there is no common approach yet to measure them [1]. As a consequence, organizations often do not measure productivity at all and trust in their intuition for optimization or adopt sub-optimal measurements, which can lead to even worse decisions or wrong incentive behavior [2].

In this study, we investigate a new approach to improve software development productivity. By definition, removing waste from a process improves productivity [3]. Waste is referred to as “*any activity that consumes resources but creates no value for customers*” [4]. However, waste is often not easy to identify, since it can be hidden behind administrative tasks, multitasking, poor prioritization,

and invisible cognitive processes [5]. This is, among other reasons, why waste identification and removal is typically approached through action research or case studies. External personnel can help a team or an organization to identify and remove waste [6–8], but this makes waste identification and quantification expensive. For that reason it is often performed as a one-time intervention, and lacking a sustainable effect on productivity improvement.

In this paper, we present a concept for a systematic yet lightweight self-reporting of waste for software development teams to guide and measure productivity improvement. We first developed a self-reporting tool for identifying and reporting waste and explored in a study whether the amount of reported wasted time is a reliable proxy to track productivity improvements in the long term. More concretely, we wanted to examine the following research questions:

**RQ1: Can a software development team identify and quantify waste by using self-reporting?** The self-reporting approach should present a good balance between weight and completeness to not miss important waste. The reporting itself should not be seen as waste but should help adding value in a sense to help improve productivity.

**RQ2: Is self-reported amount of waste correlated with software delivery performance?** To use the amount of reported waste for tracking productivity improvement we expect it to decrease as a team improves its productivity. Based on the PE-Model [1] which describes performance as a broader term including software productivity we use performance as a surrogate for productivity. To collect first evidence for this assumption we look for a relationship between delivery performance [9] and reported waste.

## 2 Related Work

“Waste” or “Muda” (Japanese for waste) has its origin in the concept of lean manufacturing. The core principle of “lean” is to eliminate non-value-adding activities, which are defined as waste. Womack & Jones proposed lean following his analysis of the Toyota Production System (TPS) [4]. TPS prioritizes waste removal by creating a culture that pursues waste identification and elimination in the entire production of a vehicle [3]. The process discerns three types of activities: activities that create value for the customer; activities that create no value for the customer but are currently necessary to manufacture the product; and activities that create no value for the customer, are unnecessary. These are considered waste and therefore should be removed immediately. Initially, the TPS characterized seven types of waste [3], which later were extended with two more waste types by Womack & Jones [4]. In their work about Lean Software Development (LSD), Mary and Tom Poppendieck adapted lean and the TPS from manufacturing to software development, and identified seven waste categories in software development [10].

Identifying waste in software development seems easy only at first glance, by potentially just observing it [5]. Some studies describe how researchers identified

and removed waste in software development teams or organizations [6–8]. All of them included external personnel which helped with the identification. Several studies have been conducted to identify and categorize waste. Sedano et al. defined a taxonomy of waste consisting of nine categories [5]. Al-Baik & Miller conducted action research to identify and eliminate waste in an IT organization and defined ten types of waste [8]. All the studies we found identified waste with a one-time intervention and applied methods with a lot of overhead, or which needed external personnel. Deshmukh and Srivastava reviewed lean methodology in software development [11], while Meyer et al. identified context switching as a universal productivity killer [12, 13]. Khodawandi categorized tasks as value add, necessary non-value add, and obvious non-value add, but found that this was not granular enough to identify wasteful activities [14]. Halkos and Bousinakis discovered that stress reduces productivity and personal satisfaction increases it [15].

Studies identified waste using Value Stream Mapping (VSM) [6, 16–18], open or semi-structured interviews [8, 19–22], analyzed content (like retrospectives) [5], conducted observations [5] or used questionnaires [20, 22, 23]. Al-Baik & Miller used three questions per waste category to identify waste [8]. Ikonen et al. did the same but used the categories defined by LSD [21]. Besker et al. investigated the time wasted caused by technical debt. They let the participants fill out a survey twice a week during seven weeks [22]. The study of Alahyari et al. used open questions but offered no help for recall [19]. Several researchers used directed questions to identify and recall waste; they derived the questions based on the waste categories. Interestingly, the well-known tool to identify waste in manufacturing, VSM, is criticized for being used in the software development context. The fundamental critique is that manufacturing is fundamentally different from software engineering. When applied in practice, participants state that the results are obvious but it takes a lot of time to apply VSM [8, 11, 17]. While there have been several approaches to capture waste in software development, all of them suffered from being either one-time interactions or heavy-weight, which reduced their sustainability. Our concept is to implement a light-weight approach for waste self-reporting which could be done on a regular basis to improve software productivity in a more sustainable way.

In recent years, the productivity aspect in software development has gained more attraction with the adoption of DevOps [9]. Besides its organizational influence, DevOps often is seen as promising to improve delivery speed, productivity, and quality [24]. Wiedemann et al. found that only four key metrics (FKM) differentiate between low, medium, high, and elite performers: lead time for change, deployment frequency, time to restore service, and change failure rate [9]. These metrics help organizations and teams to determine whether they are improving the overall IT performance. They are strongly correlated with well-known DevOps practices and hence known as DevOps metrics. However, the productivity metrics are lagging measurements and do not directly suggest any actions. Moreover, less mature teams need guidance to be able to improve on these specific metrics.

### 3 Research Method

We organized the study into three phases. In the first phase we analyzed and summarized the existing literature, theories, and frameworks about waste classifications. Then, we gathered qualitative data from the research context by performing focus group interviews with the developers from the company participating in the main study. The result was a categorization of waste enriched with context-specific elements and examples. In the second phase we addressed RQ1 “Can a software development team identify and quantify waste by using self-reporting?” and we developed a self-reporting survey and running the survey, using the waste categorizations from the preparation phase. We designed the survey by analyzing recall studies and existing waste identification studies. The survey was planned as a daily self-reporting survey for three weeks, including questions about participant’s experience with it. Finally, the third phase comprised the analysis phase, in which we analyzed the gathered data to answer RQ2 “Is self-reported amount of waste correlated with software delivery performance?” At the end of the reporting period, the participants were requested to fill out an additional survey, in which we asked them about their software delivery performance. The software delivery performance was determined using the productivity metrics suggested by Forsgren et al. [9]. Using statistical analysis, we investigated if there was a correlation with the self-reported waste collected during self-reporting survey.

The study was carried out in an IT department of a large company in Switzerland. The group consists of strategic divisions that operate in the core markets of the company and affiliated function units. Function units support the group’s management and strategic divisions with cross-cutting-concerns. In 2019 the IT function unit had around 1200 full-time equivalent employees with about 330 software developers. The unit runs only projects for internal customers. The function unit is divided into six departments. One of the departments is the department for software development and consists of 25 teams, of which 15 are software development teams. We included the software development department in the study and looked for participants for the focus groups to collect waste examples and to fill out the reporting. The resulting pool of potential participants consisted of 164 individuals, distributed over 15 teams. For self-reporting, we invited the potential participants to attend the study by e-mail. 26 employees responded to this e-mail and agreed to participate. These people represent 10 different teams from the development department. For the self-reporting (SR), twenty-two participants were included in the data analysis. Among them were two women and sixteen men, with ages between 21 and 58. Their highest degree of education is shown in Table 3, Table 2 shows the experience, and Table 4 the employment.

The focus groups (FG) aimed to get an exhaustive number of waste examples. We decided the size of one focus group to be six people and to host one session per group. Six participants per focus group and two sessions could represent twelve of the fifteen teams. To select the candidates for the focus group, we used quota sampling to ensure half of the candidates attending will also participate in self-

reporting and half will not. Moreover, we ensured that a team is represented by at most one person. Four and five participants did finally attend. Two of them were women, and the rest were men, with an average age of 35 years. Table 1 shows their experience and Table 3 their highest degree of education.

**Table 1.** Experience of focus group participants.<sup>a</sup>

Years	Working for Swiss Post	Experience Software Engineering Agile methodologies	Experience Experience	DevOps
<b>0–2</b>	5 (56%)	1 (11%)	1 (11%)	5 (56%)
<b>3–5</b>	1 (11%)	2 (22%)	3 (33%)	3 (33%)
<b>6–10</b>	2 (22%)	1 (11%)	5 (56%)	0
<b>11–15</b>	1 (11%)	3 (33%)	0	0
<b>≥16</b>	0	2 (22%)	0	1 (11%)

<sup>a</sup>Due to rounding errors, the percentages may do not sum up to 100%.

**Table 2.** Experience of self-reporting participants. <sup>a</sup>

Years	Working for Swiss Post	Experience	Experience	Experience
		Software Engineering	Agile methodologies	DevOps
<b>0–2</b>	8 (36%)	4 (18%)	5 (23%)	11 (50%)
<b>3–5</b>	6 (27%)	3 (14%)	8 (36%)	8 (36%)
<b>6–10</b>	2 (9%)	4 (18%)	9 (41%)	3 (14%)
<b>11–15</b>	4 (18%)	5 (23%)	0	0
<b>≥16</b>	2 (9%)	6 (27%)	0	0

<sup>a</sup>Due to rounding errors, the percentages may do not sum up to 100%.

**Table 3.** Highest degree of education of all study participants. <sup>a</sup>

Degree of Education	FG	SR
Vocational Education	0	1 (6%)
Higher Education	3 (33%)	2 (11%)
Bachelor of Science/BSc	4 (44%)	10 (56%)
Master of Science/MSc	1 (11%)	3 (17%)
PhD	1 (11%)	2 (11%)

<sup>a</sup>Due to rounding errors, the percentages may do not sum up to 100%.

**Table 4.** Employment of self-reporting participants.<sup>a</sup>

Employment	Number
100%	10 (56%)
90%	3 (17%)
80%	4 (22%)
70%	1 (6%)

<sup>a</sup>Due to rounding errors, the percentages may do not sum up to 100%.

## 4 Classification of Software Development Waste

We first conducted a literature review to get an initial list of empirical-based waste classifications along with examples. We then combined the found classifications to get one classification list that covered all found examples. Finally,



we held two focus group sessions to collect further examples of waste from the company and verified that the classifications also covers those examples. Table 5 lists the twelve categories used to classify the waste examples from the literature review and the examples collected during the focus group sessions.

For the literature review we used the keywords “software development”, “software engineering” and “waste”. As suggested by Gusenbauer [25], we used three search engines to retrieve literature: ACM Digital Library, ScienceDirect, and Scopus. We found sixteen empirical and peer-reviewed studies which are about waste in software engineering according to lean thinking. We extracted 105 unique examples of waste and found three studies [5, 8, 19], containing a categorization of waste. Out of the 105 examples, 85 could be assigned to a category of Sedano et al. [5]. The remaining examples could be assigned to the category “Management & organizational aspect” or “processes” from Alahyari et al. [19]. We merged those two categories as the processes are an organizational aspect. We did not use the categorization of Al-Baik and Miller [8] because the study context was a whole IT organization and not just software development.

To gather waste examples from the research context we conducted two focus group sessions [26] with the topic “How do I waste my time?” As preparation, the participants were introduced to the concept of waste by reading an instruction document prepared by the research team. We assigned each example to a waste category identified in the literature review. The examples which could not be assigned to a category were analyzed to define possibly new categories.

The first focus group session generated 77 examples from which 68 did fit into existing categories. The second session generated 90 examples, 63 could be assigned directly. 28 of the not categorized examples were about doing manual work which could have been done automatically. For example, “requesting for firewall rule changes” or “requesting for new database infrastructure”. The participants mentioned that those things need to be done by looking information up and sending e-mails within the organization. Hence, we defined an additional category, “manual work”, as “The cost of doing work manually which could be automated”. The remaining eight examples without a category were related to unreliable infrastructure and the resulting troubleshooting, the lack of responsibility and doing things that are not related to the person’s job. While those activities are necessary to reach the overall goal and hence not waste at a first glance, they distract from performing the value-adding activities. For example, troubleshooting infrastructure and platforms should not be a common concern for development teams. Therefore, we introduced the custom category “other duties” as “the cost of doing work which is supposed to be done by others.”

## 5 Waste Self-reporting Survey

The second phase comprised the development of a waste quantifiable survey questionnaire, the execution of the survey over the planned period of time, and a final retrospection survey, in which the participants reported about their experience with the waste self-reporting. To be able to quantify the reported waste,

we defined a measurement for each waste category together with its measurement unit. The identified measurements with their units are shown in Table 5 for each category and are explained in the following.

**Table 5.** Waste Categories incl. Measurement. WC11 & WC12 are new.

ID	Waste Category	Measurement and Unit
WC1	Building the wrong feature or product [5]	Customer confidence (Likert-Scale)
WC2	Mismanaging the backlog [5]	Time spent (h) & Delay (h)
WC3	Rework [5]	Time spent (h)
WC4	Unnecessarily complex solutions [5]	Time spent (h)
WC5	Extraneous cognitive load [5]	Time spent (h)
WC6	Psychological distress [5]	Stress (numerical rating scale)
WC7	Waiting/multitasking [5]	Delay (h) & Context Switches (count)
WC8	Knowledge loss [5]	Time spent (h)
WC9	Ineffective communication [5]	Time spent (h)
WC10	Management & organizational aspect [19]	Time spent (h) & Delay (h)
WC11	Manual work (new category)	Time spent (h) & Delay (h)
WC12	Other duties (new category)	Time spent (h)

*Time Spent:* The time spent is how much active time is spent on a wasteful task or activity. One example is the time spent doing rework. The time spent is measured using the unit hour.

*Delay Time:* Activities in some categories do not require active working time but cause delays. Delays are harmful because they increase the lead time. The delay is measured in hours. Delay typically occurs in combination with other observable measurements. An example is the waste category “Management & Organizational aspects.” This category is quantifiable by time spent (e.g., filling out a form for approval) and with delay (e.g., waiting for approval until one can continue with the activity).

*Stress Level:* Psychological distress is subjective to people who experience it. The Cohen Perceived Stress Scale (PSS) is widely used to measure the perceived stress. But as it is time-consuming to report. Hence, we took inspiration from pain measurement and decided to use a numerical rating scale (numerical rating scale) to measure psychological distress. On the scale zero means not stressed at all and ten extremely stressed.

*Customer Confidence:* Creating a feature that a customer does not need is a waste of time. However, if it is already evident when the feature is created that it is not needed, why is it built? In hindsight, “Building the wrong feature or product” can be quantified by the time wasted to build something; but while working on it, this may not be that clear. Hence, we decided to not measure it with the time spent, but rather with the degree of confidence a participant has, that they work on the right thing from a customer perspective with a five-point Likert scale.

*Context Switches:* The category “waiting/multitasking” can be measured by “delay time” but also by counting context switches. Interruptions or waiting which causes context switches come at a high-cost.

For the question creation, we considered several factors about how recall works. Specific questions to capture any kind of waste in a certain category possibly increases the accuracy. However, the participants report daily, and a too detailed and long survey leads to reporting fatigue. To prevent reporting fatigue, the number of questions must be minimized, and answering must be fast. To cover all categories with their measurements sixteen questions would be necessary. We excluded two categories and measurements from the survey, which finally resulted in twelve survey questions related to the categories, plus two additional open questions to get extra information.

We excluded the category “mismanaging the backlog” and a part of “waiting/multitasking” from the survey. The category “mismanaging the backlog” is very broad. It goes from “duplicated work” onto “imbalance between feature work and bug fixing” to “not enough ready stories.” Mismanaging the backlog will lead to inefficient and ineffective working manners, unnecessary time spent, and delays. Measuring this in a daily or weekly survey seems not an adequate approach for several reasons: a questionnaire with multiple questions would be necessary to identify a certain degree of mismanagement of the backlog; asking those questions seems more a topic for sprint retrospectives, rather than in short intervals. Waste of the category “waiting/multitasking” is observable at different scales. For example, multitasking can mean switching every few minutes between different tasks or working on another project every few days. This is similar for waiting. One can wait a few minutes for a build to complete or wait weeks to get formal approval to continue a task. Other categories already cover the aspects of the large scale. For the small scale, interruptions and context switches could simply be counted and there is promising research with a focus on automatically detecting and reducing harmful interruptions [13,27]. For those reasons, the category “waiting/multitasking” is not explicitly considered in the self-report survey.

We ended up with the following list of the questions asked in the survey. Beside the twelve waste identification questions, we added two optional questions: one to get qualitative information about stress and one to be able to discover waste which did not fit into our categorization.

1. How stressed did you feel today?
2. How much time did you spend on preventable rework?
3. How much time did you spend on manual or routine work?
4. How much delay (or expected delay) did you experience caused by missing automation/self-service or processes?
5. How much time did you lose because of ineffective communication?
6. How much time did you spend for unnecessary administrative/organizational demands?
7. How much delay (or expected) did you experience caused by administrative/organizational demands?

8. How much time did you spend on activities which are not your duties?
9. How confident are you that you worked on the right things today from a customer perspective?
10. How much time did you spend on unnecessary cognitive load?
11. How much time did you spend on unnecessarily complex solutions?
12. How much time did you spend because of knowledge which wasn't available?
13. Extra: What caused you stress?
14. Extra: Were there wasteful activities which you couldn't report because they did not belong to a question?

## 6 Data Collection

The data collection was conducted in three steps.

- a) An initial survey gathered participants' demographics.
- b) Then, we conducted the main survey, which went over three weeks to gather information about waste. In addition, before we started the self-reporting, two participants used the survey for two days and gave informal feedback.
- c) We concluded the data collection with a final survey to assess the completeness of the daily reporting (participants stated the days on which they deliberately did not report), to assess the usefulness of the survey tool, and to collect the productivity metrics data (see Sect. 7 details).

The software delivery performance was measured using the productivity metrics as defined by Forsgren et al. [9] and was captured from the participants at the end of self-reporting period. We took the questions and answer options from the state of DevOps report survey conducted by DORA in 2021. The daily reported waste was grouped by measurement and aggregated over the whole reporting period. The aggregation was the daily average, and the measurements were stress, time spent, delay, and customer. To get the productivity metrics score, we assigned the answers for the productivity metrics questions to a proportionally increasing score and summed them per participant. The answer "I don't know/NA" was rated with a score of zero. The worst answer (slowest/least stable) was rated with one and so on. We excluded participants who did specify "I don't know/NA" for each productivity metrics question from the data set.

The participants were asked about their subjective experiences in the final survey. We asked about three aspects of the self-reporting tool. First, if the survey helped them to recall the encountered waste. Second, if it helped identify waste, and third if the reported waste during the three-week period was representative of their usual workdays. Additionally, we wanted to know if the participants would be ready to attend repeated measurement periods.

## 7 Results

All 26 participants completed the initial survey. 22 completed at least three daily surveys of the main survey. The final survey was completed by 21 participants.

In total, 229 reports of the daily main survey were filled out. 14 out of the 22 participants filled out the survey 10 times or more over the three week period. On average, a participant reported about 4.6h of wasted time per day. Rework is the category that is reported to cause the most waste. Rework is responsible for 20% of the reported waste while the next category is other duties with 15%. For delay, one participant reported about 4 h of delay per day on average. Administrative demand caused around 5× as much delay as missing automation. In 26.5% of the time, the respondents were “completely confident” to work on the right things from a customer’s perspective. 44.2% of the respondents were “somewhat confident” and 21.9% “neutral.” Only 7.5% of the time, the respondents were “somewhat insecure” or “completely insecure” to work on the right things. Figure 1 shows the distribution. The reported stress score had a mean of 2.3, a median of 2, a standard deviation of 2.3. The minimum was zero, and the maximum was eight. The most significant stresses mentioned in free text were meetings (9×), deadlines (8×), and interruptions caused by calls or parallel tasks (4×). To further validate the categorization we gave the participants the possibility to report examples of waste which they think did not fit into one of the given categories. We found fitting categories for all eighteen examples.

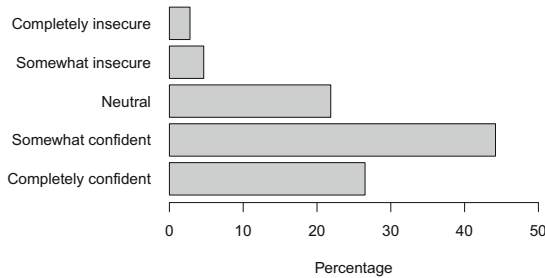
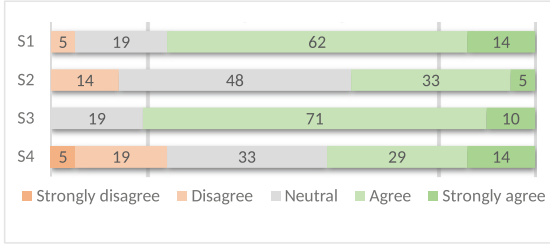


Fig. 1. Confidence to work on the right things.

**RQ1: Can a software development team identify and quantify waste by using self-reporting?** Figure 2 shows the analysis of the value of and experience with the self-reporting 76% of the participants found the survey helpful for recalling encountered waste (S1). Only 5% disagreed with that statement, and 19% were neutral about it. 38% agreed or strongly agreed that the survey helped to identifying waste, 48% were neutral, and 14% disagreed (S2). For both questions, nobody strongly disagreed. 81% of the participants agreed or strongly agreed that the reported waste during the three-week reporting period was representative of their usual workdays. 43% of the participants were willing to regularly do self-reporting, 33% expressed neutrality and only 24% disagreed with this statement. Overall, these results suggest some reason for optimism about the approach. The causes need exploration: perhaps the daily schedule was too much, or motivation, or lack of confidence were factors.



**Fig. 2.** Value of the waste self-reporting survey. S1: Helpful for recall, S2: Helpful in identifying waste, S3: Representative of usual experience, S4: Willingness to do regularly.

**RQ2: Is self-reported amount of waste correlated with software delivery performance?** 21 participants filled out the final survey with the productivity metrics questions. We excluded four of them, because they have chosen “I don’t know/NA” for every of the four productivity metrics questions. We calculated the mean daily waste reported per measurement, per participant and day. Time spent and delay are both given in mean hours per day. The participants reported using radio-buttons and selected hour ranges. We used the mean of the given range. For the time spent, it was possible to report more than eight hours per day due to the survey design. We corrected for this by adjusting two from a higher value than eight to eight. The Table 6 shows the Spearman correlation coefficient of the measures time spent, delay, stress and customer together with the productivity metrics. The measurements delay, time spent and stress show a low correlation and are not statistically significant. A visual inspection of the scatter plot does also not suggest any trends. We looked at the correlation between the productivity metrics score and the twelve waste categories without aggregating them as well but found no statistically significant correlation. Between the productivity metrics and customer confidence there is a statistically significant moderate correlation. Participants with lower productivity metrics reported a higher confidence to work on the right things from a customer’s perspective.

**Table 6.** The Spearman correlation coefficient and *p*-value for productivity metrics and waste.

	Time spent	Delay	Stress	Customer
$\rho$	-0.038	-0.27	-0.3	0.63
<i>p</i>	0.89	0.32	0.24	0.007

## 8 Discussion

### **RQ1: Can a software development team identify and quantify waste by using self-reporting?**

The self-reporting survey usage by developers indicated that it assisted the recall and quantification of software development waste. We found the survey to be less helpful in identifying new wasteful activities for the participants. We learned that three weeks of self-reporting was enough to get a representative picture of the usually encountered waste. The participants have shown a great willingness that they would participate in self-reporting regularly.

The large majority of the participants found the survey very helpful for recall and remembering encountered waste. This finding indicates that the categories and related questions are good in creating a stimulus for recalling the already known waste but are not as good in helping participants to identify waste they are not aware of. Identification of waste was found to be difficult by other researchers already [5,28]. Another aspect we found was that almost half of the participants did not fill out the daily survey at least once. This shows that a non-negligible share of the participants did not follow the process despite granular information upfront and regular customized reminders. Further data analysis would be needed to evaluate the cause for this.

The shortcomings of identification of new waste may be solved by addressing the described question-substitution effect, by a conceptual change and by providing better tooling. Due to the way cognition works, we consider providing more explanations and examples together with each question as not a promising approach. We suggest to adapt techniques of de-biasing such as asking more detailed/specific questions, changing the wording of the same question, or using nudges. However, this must be balanced with the goal of having a lightweight tool that does not burden the participants. The conceptual change may be to reverse the approach by not asking for wasteful activities per category but by asking for all activities and identifying wasteful sub-activities. The idea is similar to what Khodawandi did [14].

Using dedicated software can improve waste reporting reliability by providing support, guidance, and motivation to users. Possible elements were reliable reminders, gamification and automation. PersonalAnalytics by Meyer et al. can be a good starting point and could integrate the context-switch/interruption category [13]. Existing tools like RescueTime and ManicTime can provide inspiration. A tool could provide a list of activities for users to choose from, like Khodawandi did, and ask specific questions about each activity to identify waste. Each activity can have multiple wasteful aspects, and the questions should be based on waste categories and examples from this study.

### **RQ2: Is self-reported amount of waste correlated with software delivery performance?**

The overall results suggest that teams with high score in the productivity metrics do not report significantly less waste than teams that score low. A possible interpretation might be that a team that improves and reduces waste gets

more sensitive to waste, and hence the reported total numbers do not significantly change. That is because self-reporting waste remains subjective. This would make the absolute and relative amount of waste an unusable indicator for measuring productivity improvement. However, it could be that the productivity metrics score is not a good metric to represent the team performance. An alternative explanation is that a well-performing team, according to productivity metrics does not suffer from less waste, e.g., for organizational reasons. Or that the hypothesis only holds when looking at one team and not for comparing teams. Additionally, respondents who chose “I do not know/NA” for every productivity metrics question were excluded. But if they choose this option for one of the questions this is rated with zero points, which distorts the data.

Previous research somewhat contradicts our findings. Brown et al. found that a high amount of rework significantly differs between low, medium, and high performers [29]. However, they define rework as a combination of unplanned work and rework. Moreover, it is unclear how precise the estimations of their respondents were and if they are comparable with self-reporting waste at regular intervals as done in our study. In the state of DevOps report of 2018, Forsgren et al. found that higher software delivery performances were less prone to burnout [30]. This might be an indication that stress is reduced with a higher productivity metrics. However, it has to be considered that they compared with only three clusters of teams while we used a linear scale. Other researchers have already found that reducing waste can sometimes be straightforward [5] but also challenging when it is outside of the control of a team [31]; this is in line with the possibility that even a mature team with high productivity metrics suffers from organizational waste, like other teams with lower maturity.

## 8.1 Limitations

*Construct Validity:* The literature review comes with the limitation that it is unsure if we included all relevant studies. We used the term “waste” to retrieve articles. However, authors may have used another term in their studies to describe the phenomena of waste.

Participants read a waste document before the focus group, including a categorization example by Sedano et al. [5]. This could have caused bias and made it easier for participants to recall waste examples fitting into those categories. The moderator’s questions could have influenced the direction of the discussion.

Besides the already mentioned threats to validity for the waste example gathering, the limitations for the categorization are rooted in subjectivity. Due to timing constraints and the kind of work, we did not do researcher triangulation.

Finally, we acknowledge a deeper issue that relates to our measures but also the lean concept of “waste”: whether the categories identifies truly reflect waste in the sense of resource allocation without value. Determining this would require careful study of the decisions made by the organization and their outcomes.

*Internal Validity:* The amount of self-reported waste needs to be interpreted with caution because, as with most survey designs, responses may have been affected by the subjectivity of the respondents. The answer about the time was



given as radio buttons with the span of two hours. This methodological decision led to a loss of accuracy. Due to the pragmatic approach of self-reporting using a survey, it was possible for participants to report more than eight hours of time spent waste per day. They were advised to not report waste twice. Nevertheless, we found participants which did not always follow this rule. We corrected for this in the productivity metrics and waste correlation but not for the total reported waste. Participants missed or skipped some daily reportings. Nevertheless, the weeks with missing daily reports were included in the analysis because the visual assessment did not show that missing data led to a systematic overestimation or underestimation.

Though not relevant for the analysis, the following aspects have to be considered when interpreting the data. First, participants did not report during the same three weeks. Second, they did not all have the same degree of employment. Third, not all of them reported the whole three weeks but dropped out early. The self-reporting behavior and the answers about the self-report experience, especially the willingness to do self-reporting regularly, need to be interpreted with the consideration of selection bias. The invitation to participate in the study contained the information that it would be necessary to fill out a questionnaire every day and week.

*External Validity:* Despite reaching theoretical saturation for the categorization of waste, it is rooted in qualitative research and can not be generalized for all organizations. Another limitation lies in the low amount of data (sample size is small) and the sampling (reporting weeks not equally distributed over the year). Thus, the results of the applied statistical methods must also be interpreted with caution. Besides the methodological limitations there is a conceptual limitation. We acknowledge our approach will most likely not be able to reveal waste when it is at the core of somebody's job description but a holistic perspective is still necessary [28].

## 9 Conclusion and Future Work

The developed categorization of waste covered all waste encountered by the self-reporting participants, so appears to be a useful basis for future work in this area. Many study participants indicated readiness to engage in self-reports sessions regularly, though some expressed reservations, and in practice a number of missed reports suggests a need to explore causes – perhaps survey frequency and consequent fatigue. We speculate that the practicality of self-reporting could be significantly improved with dedicated tooling.

We did not find a significant relationship between the amount of waste and the productivity metrics. Hence, with this study we cannot provide evidence that the self-reported amount of waste decreases with improved software delivery performance. Nevertheless, we encourage doing further research and conducting a larger and longitudinal study. Future research should especially consider the limitations our study has illustrated and adjust the methodology accordingly.

More research is necessary to improve our approach, to validate our findings and to be able to draw robust conclusions. We suggest conducting future research regarding the following topics: 1) software development waste categorization should be validated in other organizations. 2) investigate how to reduce the self-reporting burden using dedicated tooling and automatic measurement. 3) replicate this study and the statistical analysis with more candidates, while addressing some of the limitations.

## References

1. Wagner, S., Deissenboeck, F.: Defining productivity in software engineering. In: Sadowski, C., Zimmerman, T. (eds.) *Rethinking Productivity in Software Engineering*, Berkeley, CA, Apress (2019). ch. 4
2. Ko, A.J.: Why we should not measure productivity. In: Sadowski, C., Zimmerman, T. (eds.) *Rethinking Productivity in Software Engineering*, Berkeley, CA, Apress (2019). ch. 3
3. Coniam, F.: A study of the toyota production system from an industrial engineering viewpoint. *Manuf. Eng.* **69**(10), 14 (1990)
4. Womack, J.P., Jones, D.T.: Lean thinking-banish waste and create wealth in your corporation. *J. Oper. Res. Soc.* **48**(11), 1148 (1997)
5. Sedano, T., Ralph, P., Peraire, C.: Software development waste. In: *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017* (2017)
6. Mujtaba, S., Feldt, R., Petersen, K.: Waste and lead time reduction in a software product customization process with value stream maps. In: *Proceedings of the Australian Software Engineering Conference, ASWEC* (2010)
7. Bufon, M.T., Leal, A.G.: Method for identification of waste in the process of software development in agile teams using lean and scrum. In: Uden, L., Ting, I.-H., Corchado, J.M. (eds.) *KMO 2019. CCIS*, vol. 1027, pp. 466–476. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-21451-7\\_40](https://doi.org/10.1007/978-3-030-21451-7_40)
8. Al-Baik, O., Miller, J.: Waste identification and elimination in information technology organizations. *Empirical Softw. Engg.* **19**(6), 12 (2014)
9. Forsgren, N., Smith, D., Humble, J., Frazelle, J.: *State of DevOps Report 2019*. Technical Report, DORA (2019)
10. Poppendieck, M., Poppendieck, T.: *Lean Software Development: An Agile Toolkit (The Agile Software Development Series)*. Addison-Wesley Professional, Boston (2003)
11. Deshmukh, M., Srivastava, P.: Literature review of lean methodology and research issues for identifying and eliminating waste in software development. In: Reddy, A.N.R., Marla, D., Favorskaya, M.N., Satapathy, S.C. (eds.) *Intelligent Manufacturing and Energy Sustainability. SIST*, vol. 213, pp. 375–388. Springer, Singapore (2021). [https://doi.org/10.1007/978-981-33-4443-3\\_36](https://doi.org/10.1007/978-981-33-4443-3_36)
12. Meyer, A.N., Fritz, T., Murphy, G.C., Zimmermann, T.: Software developers' perceptions of productivity. In: *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, vol. 16–21, November 2014
13. Meyer, A.N., Fritz, T., Zimmermann, T.: Fitbit for developers: self-monitoring at work. In: Sadowski, C., Zimmerman, T. (eds.) *Rethinking Productivity in Software Engineering*, Berkeley, CA, Apress (2019). ch. 22

14. Khodawandi, D.: Separating and quantifying value and waste to improve operational performance in software development. In: Proceedings of the 1st International Symposium on Business Modeling and Software Design (2011)
15. Halkos, G., Bousinakis, D.: The effect of stress and satisfaction on productivity. *Int. J. Prod. Perf. Manage.* **59**(5), 6 (2010)
16. Berrahal, W., Marghoubi, R.: Lean continuous improvement to information technology service management implementation: Projection of ITIL framework. In: 2016 International Conference on Information Technology for Organizations Development, IT4OD 2016 (2016)
17. Ali, N.B., Petersen, K., Schneider, K.: FLOW-assisted value stream mapping in the early phases of large-scale software development. *J. Syst. Softw.* **111**, 213–227 (2016)
18. Lehtonen, T., Kilamo, T., Suonsyrja, S., Mikkonen, T.: Continuous, lean, and wasteless: minimizing lead time from development done to production use. In: Proceedings - 42nd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2016 (2016)
19. Alahyari, H., Gorschek, T., Svensson, R.B.: An exploratory study of waste in software development organizations using agile or lean approaches: a multiple case study at 14 organizations. *Inf. Softw. Technol.* **105**, 78–94 (2019)
20. Bjarnason, E., Wnuk, K., Regnell, B.: Are you biting off more than you can chew? A case study on causes and effects of overscoping in large-scale software engineering. *Inf. Softw. Technol.* **54**(10), 1107–1124 (2012)
21. Ikonen, M., Kettunen, P., Oza, N., Abrahamsson, P.: Exploring the sources of waste in Kanban software development projects. In: Proceedings - 36th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2010 (2010)
22. Besker, T., Martini, A., Bosch, J.: Software developer productivity loss due to technical debt—a replication and extension study examining developers’ development work. *J. Syst. Softw.* **156**, 10 (2019)
23. Tuan, N.N., Thang, H.Q.: Combining maturity with agility - lessons learnt from a case study. In: ACM International Conference Proceeding Series (2013)
24. Lwakatare, L.E., et al.: DevOps in practice: a multiple case study of five companies. *Inf. Softw. Technol.* **114**, 217–230 (2019)
25. Gusenbauer, M., Haddaway, N.R.: Which academic search systems are suitable for systematic reviews or meta-analyses? Evaluating retrieval qualities of Google Scholar, PubMed, and 26 other resources. *Res. Synth. Methods* **11**(2), 3 (2020)
26. Kontio, J., Bragge, J., Lehtola, L.: The focus group method as an empirical tool in software engineering. In: Shull, F., Singer, J., Sjøberg, D.I.K. (eds.) *Guide to Advanced Empirical Software Engineering*, Springer, London, pp. 93–116 (2008). [https://doi.org/10.1007/978-1-84800-044-5\\_4](https://doi.org/10.1007/978-1-84800-044-5_4)
27. Züger, M., Meyer, A.N., Fritz, T., Shepherd, D.: Reducing interruptions at work with FlowLight. In: Sadowski, C., Zimmerman, T. (eds.) *Rethinking Productivity in Software Engineering*, Berkeley, CA, Apress (2019). ch. 23
28. Power, K., Conboy, K.: Impediments to flow: rethinking the lean concept of ‘Waste’ in modern software development. In: Cantone, G., Marchesi, M. (eds.) *XP 2014*. LNBP, vol. 179, pp. 203–217. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-06862-6\\_14](https://doi.org/10.1007/978-3-319-06862-6_14)

29. Brown, A., Forsgren, N., Humble, J., Kersten, N., Gene, K.: State of DevOps Report 2016, Puppet + DORA, Technical Report (2016)
30. Forsgren, N., Kersten, M.: DevOps metrics. *Commun. ACM* **61**(4), 3 (2018)
31. Rodríguez, P., Partanen, J., Kuvaja, P., Oivo, M.: Combining lean thinking and agile methods for software development a case study of a finnish provider of wireless embedded systems. In: *Proceedings of the Annual Hawaii International Conference on System Sciences* (2014)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# A Lean Approach of Managing Technical Debt in Agile Software Projects – A Proposal and Empirical Evaluation

Abdullah Aldaej<sup>1</sup> (✉), Anh Nguyen-Duc<sup>2</sup>, and Varun Gupta<sup>3</sup>

<sup>1</sup> Imam Abdulrahman Bin Faisal University, Dammam, Saudi Arabia  
aaaldaeej@iau.edu.sa

<sup>2</sup> University of South Eastern Norway, Notodden, Norway  
angu@usn.no

<sup>3</sup> GISMA University of Applied Sciences, Potsdam, Germany  
varun.gupta@gisma.com

**Abstract.** Technical Debt Management (TDM) includes activities such as identifying, measuring, and prioritizing technical debt. It is mainly performed to proactively mitigate the risk of losing the maintainability and evolvability of the software product which results in reducing the team velocity. Despite the importance of TDM, its adoption in software companies remain limited. Software companies are witnessing high market demand and competition that make delivering customer value outweighs the effort invested in TDM activities. Since the impacts of technical debt are uncertain and evident only in the long run, it is more difficult for companies with very limited resources to proactively spend their resources on TDM. In this paper, we propose a lean approach to facilitate the adoption of TDM in software companies with very limited resources. Based on this approach, TDM is driven by project management metrics, such as team or sprint velocity, and velocity variance. We conducted an initial evaluation of the concept of this approach through a short survey of 43 software project/product managers. Most of the survey respondents have a positive impression about our approach, which will encourage us to proceed further using more robust empirical evaluation.

**Keywords:** Technical debt · Project management · Agile software development · Sprint velocity

## 1 Introduction

Managing technical debt is reportedly non-trivial tasks for project managers and team leaders in software development projects. Technical Debt (TD) is a metaphor for a work-around technical solution that is not sustainable and requires future rework [1]. It has been shown as an important phenomenon to expedite the development in the short term in both large companies [2] and small and startup companies [3]. Technical Debt Management (TDM) is a process that includes different activities such as identifying, measuring, monitoring, prioritizing, and repaying TD. It has been adopted by some

large software companies, but at different levels of maturity [2, 4]. Although TDM is important to proactively manage the risk of TD, it introduces extra activities to the regular development which leads to extra costs [2, 5]. Consequently, many companies (especially the ones with limited resources) would find TDM difficult to adopt as part of their development process. Furthermore, it is difficult to identify and manage large instances of TD items in agile software development [6].

Managing technical debt is not independent from other aspects of project management and the project development life cycle. When it is managed, TD is often treated as a task in the product backlog along with other tasks such as new features and bug fixes. Measuring and prioritizing TD in the product backlog depends on the evaluation of its negative impact on the product. Some negative impacts of TD has been found from literature, for instance, delaying the delivery of releases [7], increasing the number of maintenance activities [8], and reducing the team productivity [9]. However, knowing the negative impact of TD is only one half of the story. Agile project managers need a precise way of measuring this impact, to better communicate it to other stakeholders.

In agile projects, there are possibilities with using agile productivity metrics, such as velocity, cycle time and takt time [10, 11] for measuring TD implications. Sprint velocity is among the most common productivity metrics in agile projects that use Scrum method [10], by showing the number of tasks (i.e., story points) that the team can accomplish in one sprint. Cycle time is defined as the amount of time that passed from when work actually started to fulfilling the work [12]. Takt time is defined as the average time interval between the start of development of two sequential software versions to meet customer demand [13, 14]. This measure helps to identify if the software team can meet the customer demand, exceeding the demand (over producing) or unable to meet demand (under producing). To the best of our knowledge, there is no study that connects the agile team productivity metric with the TDM process.

In this paper, we propose a lean approach for TDM by incorporating the usage of agile productivity metrics within the TDM process. This lean approach is used specifically for TDM within the agile software development methodology. In general, lean is an approach that focusses on delivering maximum value with minimum waste. Our approach is “lean” in a way that emphasizes on minimizing the effort and complexity of TDM, by focusing on monitoring and addressing TD implications quantitatively. This can facilitate the adoption of TDM by 1) providing a lightweight approach for TDM in agile software projects, and 2) bridging the communication gap between technical and non-technical people for decisions related to allocating resources for TDM. We conducted a short survey of 43 software product and project managers to initially evaluate the concept of our approach. The results reveal that most project managers foresee potential advantages of our approach.

## 2 The Lean Technical Debt Management Process

### 2.1 The Management Approach

The proposed approach introduces a new paradigm of dealing with TD. Based on this approach, TDM is driven by a popular metric in agile project management such as sprint velocity. In addition to its usage for measuring the development velocity, we use this

metric as a quantitative measure for TD implication. Since dealing with TD involves high uncertainty about its future impacts (i.e., difficulty to measure TD interest), TDM is proceeded based on a control threshold of the development velocity.

Our approach applies a lean mechanism that simplifies the way of dealing with TD. It assumes that issues that directly affect the development velocity are considered TD issues. This assumption is based on evidence from previous studies that TD can be manifested in multiple aspects (beyond software code and architecture) such as process, people, social [11, 15]. Also, TD related to process and people are found to be more interested to project management practitioners [16]. In addition, the development speed (i.e., delivery speed) is identified as the most common effect of TD [7].

Figure 1 shows our lean approach for TDM. It connects some project management activities (PM layer) with TDM activities (TDM layer). At the PM layer, the agile project managers monitor the team velocity at the end of each sprint using some metrics (e.g., sprint velocity) which is usually available in the PM data. In addition, the project managers need to establish control thresholds for the velocity that determine the optimal velocity range. At the end of each sprint, the project managers check the variance between the actual and optimal sprint velocity. If the actual velocity is below the optimal, then the project managers can measure the impact of such low velocity by calculating the difference (i.e., TD interest). The following formulas are used to measure velocity and TD interest:

$$\text{Actual } (V) = \text{Actual velocity of the current sprint} \quad (1)$$

$$\text{Optimal } (V) = MD(\text{optimal velocity range}) \quad (2)$$

$$\text{Current } (TDI) = \text{Optimal } (V) - \text{Actual } (V) \quad (3)$$

$$\text{Future } (TDI) = \text{Current } (TDI) * \# \text{Planned release} \quad (4)$$

where  $V$  indicates Velocity,  $MD$  refers to Median,  $TDI$  stands for TD Interest, and  $\#$  is the Number of future released to be considered in the prediction.

Since our approach is driven by TD implication (which is measured using the development velocity), It begins by measuring TD interest before identifying TD items. The TD interest is precisely measured for the present as the difference between current and optimal velocity (step 2.1 in Fig. 1). After that, the TD interest can be estimated in the future considering the most recent TD interest and the number of future releases/sprints that captures the future timeframe (step 2.2 in Fig. 1). This information can help project managers to communicate with non-technical stakeholders (e.g., high management or investors). It also supports decision making related to allocating resources for TDM. If the non-technical stakeholders decided to spend resources for TDM, then the project managers can begin the process of TDM (i.e., move to the TDM layer). If not, then the project managers will remain on the PM layer and continue 1) monitoring the development velocity, and 2) informing the non-technical stakeholders about the velocity variance and its related TD impacts.

TDM layer includes the main TDM activities [4], which are proceeded after the agreement to allocate resources for TDM. This layer includes some TDM activities

such as TD identification (step 3 in Fig. 1), TD measurement (step 4 in Fig. 1), TD prioritization (step 5 in Fig. 1), and TD repayment (step 6 in Fig. 1). The TDM layer is adopted from the TDM frameworks in [4, 17]. However, we eliminate activities such as TD monitoring and TD prevention since our approach concentrates on monitoring TD implications (not TD items). Also, our approach applies a different mindset for TDM that is driven by a control threshold of TD implications (with less emphasis on TD prevention). In the TDM layer, the project managers begin the TDM process by identifying TD items which are (according to our approach) the causes that are directly associated with the velocity reduction. Then, TD items are measured as the cost of fixing each item (i.e., TD principal). After that, TD items can be prioritized based on their fixing cost and their impact on the development velocity. Finally, TD repayment is performed starting from TD items with highest priority.

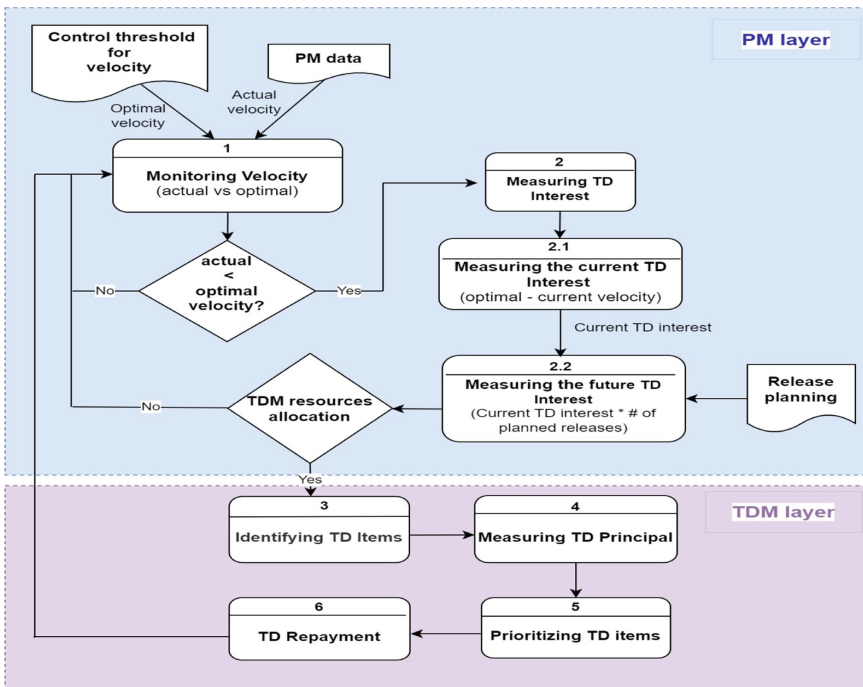


Fig. 1. Lean TDM process

## 2.2 Use Scenarios

Table 1 shows an example of how the lean TDM approach can be applied. In this example, we classified the development velocity into three risk levels: low risk, moderate risk, and high risk. The low-risk indicates the optimal speed range based on the company’s goal. At this level (low risk), the team can confidently skip TDM. It means that it is not efficient to allocate resources for TDM since no symptoms appears. When the speed is






at the moderate-risk range, it is an indication for early TD symptoms. At the moderate risk level, the team needs to keep non-technical stakeholders informed about this early TD symptoms and their impact by calculating the TD interest as the difference between current and optimal/planned speed. This high-level of measuring TD interest considers the observed (fact) consequences of TD that is the gap in the development speed. In case the optimal speed is determined based on a range of value, the median value of the optimal range can be used to compare with the actual value.

In addition, the lean TDM approach can be used to provide a guidance (recommendation) for decisions to allocate resources for TDM. For example, at optimal velocity range, the recommendation would be spending resources on TDM is not mandated. When reaching the moderate range, it is recommended to allocate resources for TDM, but TD repayment is optional or can be partially implemented. At the low velocity, TD repayment is needed. Based on this example, the team should not wait until reaching the low velocity. But rather proactively informing the high management about TD consequences when reaching the moderate range, using precise fact-based measures of TD interest.

The example in Table 1 is used only to illustrate one way of applying our approach. But it is flexible for any context to select its own metric and control threshold for the development velocity. At the end, the overall concept is the same that to have a common control threshold for the development velocity that is applied within the team, and such a threshold drives the TDM process and TD interest measurements.

**Table 1.** Lean TDM Approach at PM Layer – An Example.

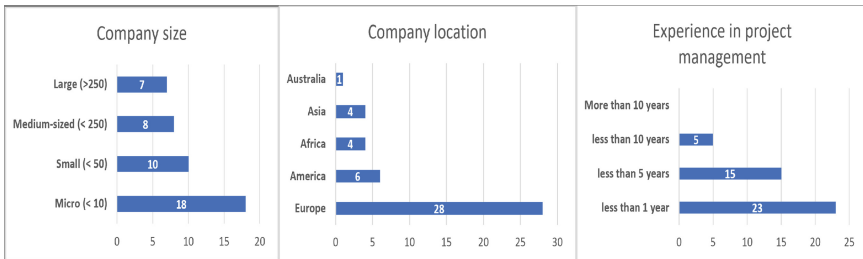
The dev speed	Risk level	Current TD interest	Future TD interest	Guidance for effective TDM resource allocation
 Optimal speed	Low risk	--	--	No resources should be spent on managing or repaying TD
 Moderately deteriorated speed	Moderate risk	Equation (3)	Equation (4)	The team should allocate some resources for managing TD
 Highly deteriorated speed	High risk	Equation (3)	Equation (4)	The team should allocate some resources for managing and repaying TD

### 3 Empirical Validation

To initially validate our proposed lean TDM approach, we surveyed 43 software project/product managers from different software companies around the world. We targeted software project/product managers because our approach focusses on the project management activities. For example, seeking resources for the product from other stakeholders is the main role for the project/product managers. Sampling of this population was performed using convenience sampling method [18]. In addition, we recruited some participants (22 out of 43) from a professional recruitment channel, which is Prolific<sup>1</sup>.

The survey instrument is available in our supplemental material<sup>2</sup>. It is a short survey that primarily intended to communicate and get feedback about our lean TDM approach. It consists of three sections. The first section presents a brief explanation about our idea to participants, and shows a table that illustrates our idea. The second section includes some questions that characterize the participants, such as company location, company size, and years of experience in the project/product management role. Finally, the third section asked respondents to initially evaluate our idea based on three technology acceptance metrics from TAM (Technology Acceptance Model) [13], which are perceived ease of use, usefulness, and predicted future use. Before disseminating the survey, we validate the survey internally that each author read the survey's questions and provide his feedback. At the end, we conducted a consensus session where we discuss the comments and reach consensus about the survey.

A total of 43 participants filled the survey. Figure 2 shows the characteristic of our participants. Most of the participants works in companies in Europe with less than 5 years of project management experience. In terms of company size, our participants work in different sizes of company, but most of them in small companies.



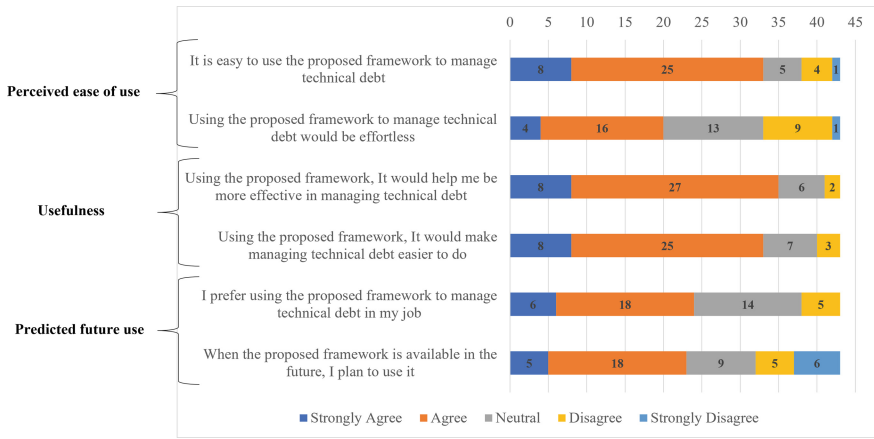
**Fig. 2.** Characteristics of participant

Figure 3 summarizes the participants' opinion on our proposed approach. It presents their answers to the closed questions in the third section of the survey (TAM related questions). On average 61.6% of the respondents perceived that the proposed approach would be easy to use (strongly agree or agree). In assessing the usefulness, on average 79.05% of the respondents strongly agree or agree that the proposed approach would

<sup>1</sup> <https://prolific.co/>.

<sup>2</sup> <https://bit.ly/3JCWQzy>.

be useful for project managers. Finally, on average 54.7% of the respondents strongly agree or agree that they will use the proposed approach in future to manage TD.



**Fig. 3.** Results of the initial evaluation to lean TDM approach

Some participants reported in the open question that the proposed approach is more helpful for an organization with very limited resources. For example, small organizations or startups usually do not have dedicated quality assurance roles. This kind of organization often relies on external support (e.g., consultancy agency) when such quality assurance is required. However, other participants were a bit skepticism as the actual usefulness and ease of use could only be predicted after using the proposed process. The dataset is available in our supplemental material<sup>3</sup>. In sum, this initial results provides early insights into the potential usefulness of our approach by software project managers.

## 4 Discussions

Previous work proposed different TDM frameworks and strategies to encourage the use of TDM in practice [19, 20]. Other studies introduced different techniques to support software teams to perform specific TDM activities, such as TD identification [21], TD measurement [22], TD prioritization [23]. However, most of the previous studies focus on TDM from certain dimensions (e.g., code, design, and architecture, etc.). Since TD is a multidimensional phenomenon, it can be difficult to have a framework that manages all aspects of TD. What distinguish our paper from previous work is that it addresses the multidimensionality of TD by increasing its level of abstraction. That we consider TD as issues that are associated with the development speed. We think that this approach can simplify the adoption of TDM in practice. It can also amplify the communication bridge between TDM and agile PM activities.

<sup>3</sup> <https://bit.ly/3Y0PJ8J>.

## 4.1 Threats to Validity

There are four main limitations in our approach. First, we assume that all issues contributed to the reduction of development velocity are considered TD. Although this assumption might simplify the adoption of TDM, it can introduce a risk of including some issues not related to TD. To address this risk, we plan (in our future work) to add a risk factor in our equation to account for this risk. Second, our approach depends on existing metrics that are used to measure team velocity. We assume that project managers have an existing metric in place related to team velocity. However, a metric such as team/sprint velocity is found to be a common one in agile software project management [10] which can minimize this threat in the agile software development context. Third, the process of identifying TD items (step 3 in Fig. 1) depends on people experience and how thoughtful they investigate the issues that cause the development speed reduction. So, the quality of TD identification depends on people experience. Lastly, we recruited some participants using Prolific, a professional channel for research participant recruitment. Despite the limitation for its lack of explicit design, it has emerged as a viable and cost-efficient option for researchers [24].

## 5 Conclusions

The application of TDM in practice remains limited especially in organizations with limited resources (i.e., those organizations that do not have a mature quality assurance team). In this paper, we proposed a lean approach to facilitate the adoption of TDM in such organizations. Our approach introduces a new mindset for managing TD that is driven by TD implications/symptoms (primarily the velocity of development). It introduces an alternative way of measuring TD interest, which is based on fact-based consequences of TD (i.e., the gap between the actual and planned development speed). We sent a short survey to software project/product managers to get their initial feedback about our approach. The majority of them have positive expectations for our approach. For future work, we plan to perform a longitudinal case study, which includes interviews at different points of time, to empirically validate and refine our approach.

## References

1. Kruchten, P., Nord, R.L., Ozkaya, I.: Technical debt: from metaphor to theory and practice. *IEEE Softw.* **29**(6), 18–21 (2012). <https://doi.org/10.1109/MS.2012.167>
2. Martini, A., Besker, T., Bosch, J.: Technical debt tracking: current state of practice: a survey and multiple case study in 15 large organizations. *Sci. Comput. Program.* **163**, 42–61 (2018). <https://doi.org/10.1016/j.scico.2018.03.007>
3. Cico, O., Souza, R., Jaccheri, L., Nguyen Duc, A., Machado, I.: Startups transitioning from early to growth phase - a pilot study of technical debt perception. In: Klotins, E., Wnuk, K. (eds.) *ICSOB 2020. LNBIP*, vol. 407, pp. 102–117. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-67292-8\\_8](https://doi.org/10.1007/978-3-030-67292-8_8)
4. Yli-Huumo, J., Maglyas, A., Smolander, K.: How do software development teams manage technical debt? – an empirical study. *J. Syst. Softw.* **120**, 195–218 (2016). <https://doi.org/10.1016/j.jss.2016.05.018>

5. Guo, Y., Seaman, C., da Silva, F.Q.B.: Costs and obstacles encountered in technical debt management – a case study. *J. Syst. Softw.* **120**, 156–169 (2016). <https://doi.org/10.1016/j.jss.2016.07.008>
6. Holvitie, J., et al.: Technical debt and agile software development practices and processes: an industry practitioner survey. *Inf. Softw. Technol.* **96**, 141–160 (2018). <https://doi.org/10.1016/j.infsof.2017.11.015>
7. Ramač, R., et al.: Prevalence, common causes and effects of technical debt: Results from a family of surveys with the IT industry. *J. Syst. Softw.* **184**, 111114 (2022). <https://doi.org/10.1016/j.jss.2021.111114>
8. Kruchten, P., Nord, R., Ozkaya, I.: Managing Technical debt - Reducing friction in software development. in *SEI Software Engineering*. Pearson Education (2019)
9. Besker, T., Martini, A., Bosch, J.: Software developer productivity loss due to technical debt - a replication and extension study examining developers' development work. *J. Syst. Softw.* (2019). <https://doi.org/10.1016/j.jss.2019.06.004>
10. Kupiainen, E., Mäntylä, M.V., Itkonen, J.: Using metrics in agile and lean software development – a systematic literature review of industrial studies. *Inf. Softw. Technol.* **62**, 143–163 (2015). <https://doi.org/10.1016/j.infsof.2015.02.005>
11. Malakuti, S., Heuschkel, J.: The need for holistic technical debt management across the value stream: lessons learnt and open challenges. In: 2021 IEEE/ACM International Conference on Technical Debt (TechDebt), pp. 109–113 (2021). <https://doi.org/10.1109/TechDebt52882.2021.00021>
12. Budacu, E.N., Pocatilu, P.: Real time agile metrics for measuring team performance. *Informatica Economica* **22**(4) (2018)
13. Taghizadegan, S.: Design for lean/kaizen six sigma. In: Taghizadegan, S. (ed.) *Essentials of Lean Six Sigma*, pp. 59–101, Butterworth-Heinemann, Burlington (2006). <https://doi.org/10.1016/B978-012370502-0/50008-4>
14. Thollander, P., Karlsson, M., Rohdin, P., Wollin, J., Rosenqvist, J.: 14 - energy management using lean. In: Thollander, P., Karlsson, M., Rohdin, P., Wollin, J., Rosenqvist, J. (eds.) *Introduction to Industrial Energy Efficiency*, pp. 259–287. Academic Press (2020). <https://doi.org/10.1016/B978-0-12-817247-6.00014-6>
15. Martini, A., Stray, V., Moe, N.B.: Technical-, social- and process debt in large-scale agile: an exploratory case-study. In: Hoda, R. (ed.) *XP 2019. LNBP*, vol. 364, pp. 112–119. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-30126-2\\_14](https://doi.org/10.1007/978-3-030-30126-2_14)
16. Gomes, F., dos Santos, E.P., Freire, S., Mendonça, M., Mendes, T.S., Spínola, R.: Investigating the point of view of project management practitioners on technical debt - a preliminary study on stack exchange. In: 2022 IEEE/ACM International Conference on Technical Debt (TechDebt), pp. 31–40 (2022). <https://doi.org/10.1145/3524843.3528095>
17. Seaman, C., Guo, Y.: Measuring and monitoring technical debt. In: *Advances in Computers*, p. 22 (2011)
18. Baltes, S., Ralph, P.: Sampling in software engineering research: a critical review and guidelines. *Empir. Softw. Eng.* **27**(4), 94 (2022). <https://doi.org/10.1007/s10664-021-10072-8>
19. Nikolaidis, N., Zisis, D., Ampatzoglou, A., Chatzigeorgiou, A., Soudris, D.: Experience with managing technical debt in scientific software development using the EXA2PRO framework. *IEEE Access* **9**, 72524–72534 (2021). <https://doi.org/10.1109/ACCESS.2021.3079271>
20. Wiese, M., Rachow, P., Riebisch, M., Schwarze, J.: Preventing technical debt with the TAP framework for technical debt aware management. *Inf. Softw. Technol.* **148**, 106926 (2022). <https://doi.org/10.1016/j.infsof.2022.106926>
21. Tu, H., Menzies, T.: DebtFree: minimizing labeling cost in self-admitted technical debt identification using semi-supervised learning. *Empir. Softw. Eng.* **27**(4), 80 (2022). <https://doi.org/10.1007/s10664-022-10121-w>

22. Avgeriou, P., et al.: An overview and comparison of technical debt measurement tools. *IEEE Software* (2020). <https://doi.org/10.1109/MS.2020.3024958>
23. Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., Fontana, F.A.: A systematic literature review on technical debt prioritization: strategies, processes, factors, and tools. *J. Syst. Softw.* **171**, 110827 (2021). <https://doi.org/10.1016/j.jss.2020.110827>
24. Palan, S., Schitter, C.: Prolific.ac—a subject pool for online experiments. *J. Behav. Exp. Financ.* **17**, 22–27 (2018). <https://doi.org/10.1016/j.jbef.2017.12.004>

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# An Empirical Study About the Instability and Uncertainty of Non-functional Requirements

Luiz Viviani<sup>1</sup>, Eduardo Guerra<sup>2</sup>, Jorge Melegati<sup>2</sup>,  
and Xiaofeng Wang<sup>2</sup>

<sup>1</sup> Institute for Technological Research, São Paulo, Brazil

<sup>2</sup> Free University of Bozen-Bolzano, Bolzano, Italy

{eduardo.guerra, jorge.melegati, xiaofeng.wang}@unibz.it

**Abstract.** Managing non-functional requirements (NFRs) has been a challenge in software development for many years. These requirements are typically used to make important architectural decisions early in the project, which can be problematic if they are uncertain or unstable. When this uncertainty is not considered when designing the software architecture, changes are often costly and sometimes even unfeasible. Some empirical studies on the subject have already been carried out, but few have focused on the perspective of professionals with extensive experience on the changes and uncertainties of NFRs. This work aims to expand the understanding about the management, clarity and validation of NFRs to fill this gap in the literature. To achieve this goal, a survey was carried out with professionals to find out how NFRs were managed and validated. For the research design, instead of generic questions, the questionnaire focused on some specific types of NFRs to induce participants to recall and report concrete situations. As a result, 40 valid responses were obtained, most from professionals with more than 10 years of experience. The results reveal that a significant number of NFRs were defined after the delivery of software increments (more than 30%) and that revision and change occurred in about a third of the NFRs. Hence, this study presents evidence that NFRs, as the functional ones, can also be uncertain and change frequently, requiring agile approaches and techniques to evolve the software architecture to consider this uncertainty.

**Keywords:** Non-functional requirements · quality attributes · software maintenance · software evolution · requirements engineering · empirical study

## 1 Introduction

Non-functional requirements (NFRs) are considered an important factor in software architecture decisions. However, the most important architectural decisions are usually made in the beginning of the project, even on agile projects [38], when some NFRs are uncertain. Since changes in software architecture are often expensive when its structure is not prepared to handle them, a late definition of these requirements might represent a big risk for the project [24].

© The Author(s) 2023

C. J. Stettina et al. (Eds.): XP 2023, LNBIP 475, pp. 77–93, 2023.

[https://doi.org/10.1007/978-3-031-33976-9\\_6](https://doi.org/10.1007/978-3-031-33976-9_6)

The uncertainty of functional requirements (FRs) is handled by agile methods by using techniques that allow the code to be more easily changed, such as TDD and refactoring [17]. During the early phases of software development, it is common to see FRs being prioritized while NFRs are ignored or neglected until later stages [6]. However, agile techniques, such as TDD and refactoring, might not be enough for changes in NFRs, which can crosscut functionalities and affect the entire system. Changing or introducing an NFR that is not compatible with the current architecture might turn a software project into a failure [24].

Most NFRs are often handled ad-hoc during system testing, and software engineers pay more attention to functional requirements that satisfies business needs [29]. This neglect or late focus might affect the final product, as both FRs and NFRs are necessary for the systems' success [35]. Instability, uncertainty and shallow focus on NFRs can result in high long-term maintenance costs [9]. Although the relevance of NFRs is widely accepted, the discourse and approaches on how to deal with them are still dominated by how to differentiate them from functional requirements [11, 16]. Although more effort has been invested in satisfying NFRs [5], it seems that there is still an uneven emphasis on the importance given to the system's features and its quality requirements [15]. Understanding when NFRs are defined and how they change during the project can provide valuable evidence to guide software development, in particular, how the solution design and architecture accommodate these changes.

Some empirical studies on NFRs are present in the literature, but few focusing on professionals with large experience and high involvement in NFR management. The studies define or evaluate artifacts for the documentation of NFRs [32], compare methodologies and practices of traditional and agile requirements engineering [1] or even propose new artifacts or automation in document generation [30, 32]. Other studies [20, 24, 26, 34] aimed at identifying flaws in NFRs definition and management, but they do not discuss the validation of these NFRs, nor the impacts and difficulties caused by changes during development.

This study aims to gather evidence on the late definition of NFRs, either due to a lack of identification at the project start or due to later changes. That evidence highlights the need for more agile approaches to deal with changes and late identification of NFRs, since the ones used to handle changes in the functional requirements do not consider that it might have a huge impact in the software architecture. To achieve this goal, we conducted a survey with professionals with experience managing NFRs. The questionnaire focused on specific NFR types, stimulating participants to base their answers on concrete situations rather than on general impressions. As a result, evidence was identified of a relevant number of NFRs being defined after the start of the development and changed in late-stages of development.

## 2 Background and Related Work

NFRs refer to system restrictions and the quality of their development and implementation, which is always related and connected to FRs [28]. The IEEE defines



a non-functional requirement as “a software requirement that describes not what the software will do, but how the software will do it” [19]. In other words, NFRs are related to quality standards, as they describe the characteristics and needs of software so that it is efficient and fits its purpose [22]. The International Organization for Standardization (ISO) defines software quality as the level of satisfaction with the stated and implied needs of the user [18]. Software quality can be divided into two views: the first being that of “meeting specifications” and the second being “satisfying the customer”, the latter means satisfying the customer’s usage and consumption expectations, regardless of any measurable characteristic [18]. NFRs can be related to both observable attributes, such as performance, availability, and reliability of the system, as well as relative internal characteristics, for example, maintainability and portability [14, 16].

Since NFRs express objectives that may conflict with each other, these requirements must be described as flexible goals. These conflicting objectives lead to a need of refinement and negotiation to reach acceptable solutions [13]. According to Berntsson et al. [9], the perception of the importance and purpose of an NFR may differ according to the professional’s role in software development.

Several studies present the importance of NFRs to software project success. For instance, when NFRs are incorrectly managed, software projects have a failure rate of 60% or more [6]. NFRs play a critical role during software life cycle, and if defined or managed incorrectly, affect software maintainability, significantly impacting deadlines and costs [13]. NFRs generally influence the system architecture more than FRs [4, 31], being key to ensuring a proper architecture.

Albeit their importance, NFRs receive less attention than their functional counterpart, causing customer dissatisfaction and rework and impacting time and cost [34]. This issue happens especially in projects employing agile methods [30], given these practices emphasis on functional aspects and lesser in documentation. The neglect of NFRs is a known issue for agile requirements engineering [20]. This issue is linked to agile methods’ main characteristics of accepting and absorbing changes during the project and a lack of information at the beginning of the project for a proper upfront architecture [24]. However, other agile approaches such as Scaled Agile Framework (SAFe) and Disciplined Agile Delivery (DAD) have steps and procedures that aim to anticipate and mitigate the risks associated with RNFs and their volatility [2, 27].

Given the possibility of architectural changes, early decisions on architecture in agile projects are encouraged [3, 38, 40]. This early definition contrasts with the recurring changes in agile projects, especially in the NFRs, which mostly cause impacts or revision in the architecture [24]. For instance, Ameller et. al [4] reinforce the importance of continuous management of the NFRs, stating that the list of NFRs of the project could never be considered complete even after the completion of development tasks. According to this work, this list should be evolved and negotiated during all project development and maintenance phases. The emphasis on continuously managing NFRs, reviewing and adapting the architecture to cope with possible impacts can be identified in the Sprint Zero of the

DAD [2] approach and in the Architectural Runway phases of the SAFe framework [27], as well as all necessary adjustments for the proper testing strategy for these requirements [2].

Some studies present evidence of the awareness about the importance of NFRs, but without adequate focus on their management [10]. Other studies evidence that lower importance is given to NFRs compared to FRs during software development [36], which explains the significant deficits in the management of NFRs, and how they are treated and prioritized. Only 40% of the NFRs are measurably specified [4, 31], which makes hard its verification and assessment. The unified management of FRs and NFRs is defended by some studies, e.g., [8], which describe them as orthogonal. Consequently, architectural decisions that accommodate an FR may conflict with an NFR or vice versa.

### 3 Research Design

There is a lack of studies in the literature on the stability of NFRs, in particular, on when and how often they are defined or modified. Thus, as the objective of this study, we collected evidence from real projects on the definition and possible modification of NFRs. For example, we tried to understand in which project steps NFRs are defined and if they are modified in stages that lead to greater impact. To guide the research, we proposed the following research questions:

- RQ1** - How are NFRs defined in the software development process?
- RQ2** - How do NFRs change and are updated during a software project?
- RQ3** - How much does the approach for handling NFRs vary based on its type?

#### 3.1 Methodology

To answer the research questions, we conducted a survey with professionals consisting of questions about the occurrence of specific types of requirements in real projects. The survey was designed to be answered by professionals with experience managing NFRs in software development projects. When answering the questions, participants should report if they recall the presence of a particular type of requirement in a project they had participated. In the case of a positive answer, a series of questions about this occurrence is presented, focusing on understanding when it had been specified, if it had changed, and why it changed.

As a first validation step, a first version of the questionnaire was created and reviewed by the authors and other invited collaborators. The survey was refined and sent to five professionals as a pilot study. Based on feedback from respondents, the demographic questions were improved, new options were added, and examples of NFRs were used to make them easier to understand. The result was the final version of the survey applied in the study. The participants from the pilot study were invited to answer the questionnaire again.

As a dissemination strategy, the authors sent invitations for participation to personal contacts, to software development email discussion groups and shared them on social media. It was intended for more experienced professionals and

that the estimated time to respond was 20 min, but not many spontaneous responses were received. Most participants were professionals with the target audience’s profile, recruited through a direct invitation from one of the authors. The described approach for convenience sampling aligns with some guidelines from Rainer and Wohlin [33] to recruit credible participants.

We acknowledge that our sampling approach does not provide a representative sample (c.f. [7]). However, our goal could be stated as exploratory, i.e., looking for evidence of problems regarding NFR management and not to describe, for instance, how prevalent these problems are. We read and interpreted the answers to the open questions considering the answers given to the multiple choice questions, searching for any relevant complementary information.

### 3.2 Survey Design

Kitchenham and Pfleeger [23] suggest that the survey design must relate to research objectives so that the data obtained and the analysis can answer the questions defined for the study. We adopted a descriptive research design to conduct a retrospective study, where participants report past events and circumstances to support the understanding and explanation of current phenomena [23].

The questionnaire is available online<sup>1</sup> and was organized into eleven steps: (1) explanation of the research goals, presentation of the information about data confidentiality, and request of participant’s consent; (2) presentation of the NFR concept adopted in the study and a request to choose a project in which the respondent participated as a reference for the following answers; (3) questions about participant background and the company in which the project used as a reference took place; (4) questions about the adopted practices for requirements engineering; (5–10) for each particular NFR type, a question regarding the existence of that particular type, and, in case of a positive answer, additional questions about it; (11) an optional open question about how NFR uncertainty was handled in the reference project.

The second step was included to make sure that the participant chooses a project as a reference for gathering data about a concrete experience. The participant can only proceed after confirming that the choice was made. To avoid misconceptions about the term “non-functional requirement”, a short description was presented to the participant to agree for continuing.

In the third step, about the company, we asked the business segment and the number of employees. About the participants, the survey asked which role they played in the last five years (being able to select more than one option), years of experience in software development, and self-evaluation about the knowledge on NFRs. About the self-evaluation, five levels were defined (novice, advanced beginner, competent, proficient, and expert) with a description describing the expected expertise and knowledge for each. The fourth step assessed some information about how the reference project handled NFRs. The participant answered

---

<sup>1</sup> <https://www.jmelegati.com/files/Survey-NFRs.pdf>.

about the level of influence different roles have in defining NFRs, and how often these requirements were described in a more quantitative and measurable way or using a more qualitative and abstract approach.

In steps 5–10, each section focused on a particular NFR type. The decision to ask about specific NFRs was for the participants to recall a concrete past experience in which they could answer the questions more precisely. Limiting to six types enabled the exploration of different types of NFR without doing the survey very long. The chosen NFRs were: response time, number of simultaneous requests, scalability, flexibility, portability, and robustness. For each type, a definition and several concrete examples were provided to avoid misunderstandings.

If the respondent confirmed the presence of that type of NFR in the reference project, the following multiple-choice questions were presented: (a) approach used for NFR elicitation; (b) during each software development activity it was elicited; (c) if it was reviewed or modified during the project; (d) in case of a positive answer in the previous question, the reason it was reviewed or modified; (e) the current adherence of the NFR to the current project state. Finalizing each step, an optional open question asks details about the reported NFR.

The choice of these six specific types of NFRs can be considered a decision in the study scope and was motivated by the authors' individual experiences in industry in the field of software architecture. In the discussions conducted during the survey design, they pointed out that they had observed recurrent changes and instability on these types of NFRs. We recognize that this choice might introduce a bias in the study and we understand that the results should be interpreted as restricted to them. However, the six NFR types represent different kinds of quality attributes and cover relevant types of NFRs. Future surveys might replicate this study with other NFRs and compare the results.

## 4 Results

The survey received 40 answers. The pilot was executed in December 2021, and the survey received responses from April to June 2022. The following sections describe the obtained results.

### 4.1 Participant and Company Profile

Among the 40 professionals who answered the survey, more than 50% said they had worked in the last five years as a software architect or software engineer, and almost 40% as a developer. Other roles receiving a significant number of answers were researcher (20%), product manager (17%), and requirements analyst (17%).

The respondents fit the target profile based on the answers assessing their professional experience. Figure 1 presents the distribution of the number of participants according to their years of experience and their self-evaluation of their expertise in handling NFRs in projects. Most answers came from experienced professionals in the field.

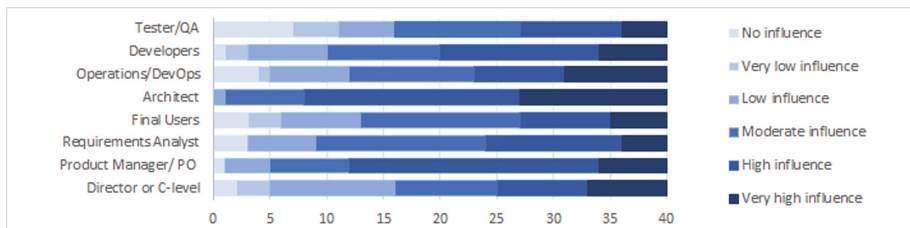
The results revealed that projects used as a reference were developed in companies from diverse segments, emphasizing the financial and e-commerce

NFR Experience	1 to 3 years	4 to 6 years	7 to 9 years	10 to 12 years	13 to 15 years	15 years or more
Novice	0	1	0	0	0	0
Advanced Beginner	0	0	0	0	0	1
Intermediate	1	2	1	5	1	6
Proficient	0	0	2	1	4	6
Specialist	0	0	1	1	0	7

**Fig. 1.** Number of participants by years of experience and self-evaluated knowledge in handling NFRs

branches, which together represent 43% of the companies. However, we also received answers from 16 other segments, which received from one to three answers (several informed in the “Other” option). About the company size, the ones with more than 5000 employees add up to more than 56%. Since companies with a range from 500 to 2000 employees were declared by 18%.

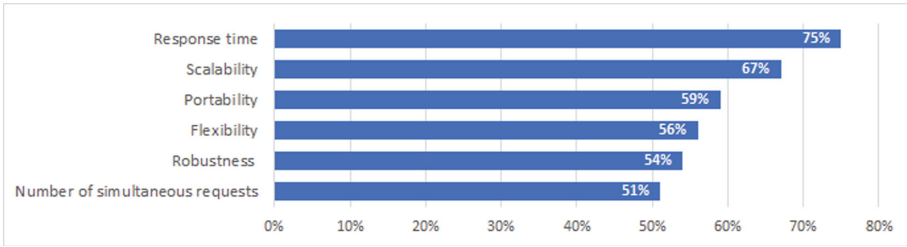
Figure 2 presents the distribution of the answers about the influence of different roles on NFRs definition. Architects, product managers, and developers are among the roles with more significant influence. On the other hand, the ones less influential were director/c-level, tester/QA, and final users. In the question in which the respondents needed to classify from 1 to 5 how often they adopt a qualitative versus a quantitative approach for NFR specification (where one was always qualitative and five was always quantitative), the result was the following distribution: 1 (10%), 2 (40%), 3 (23%), 4 (23%), and 5 (4%).



**Fig. 2.** Influence of Different Roles on NFRs

## 4.2 Answers About NFRs

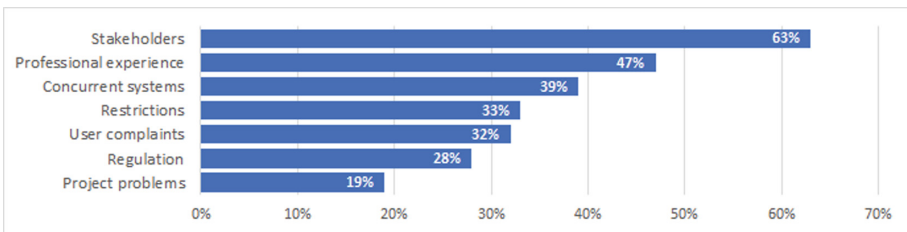
As justified in the survey design, to obtain answers based on concrete situations from real projects, the questions concentrated on six types of NFRs. Figure 3 presents the percentage of answers reported to have each NFR type present in the reference project.



**Fig. 3.** Proportion of the projects in which the type of NFR was declared

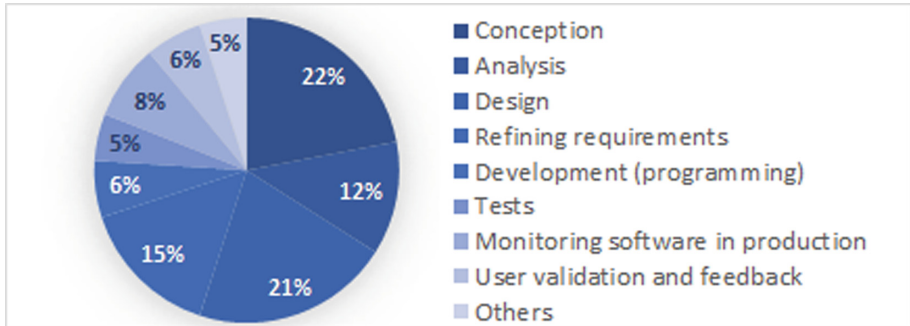
For each NFR declared as present in the reference project, the survey asks for information about the method used for elicitation, time of definition, constancy during development, possible change motivators, and validation of project adherence to the defined NFR. A general analysis of the NFRs present in the study presents relevant information about the treatment and management of the NFRs. To perform this analysis, each requirement reported independent was considered of its type, which resulted in 144 instances. To answer RQ1 and RQ2, we considered these 144 instances together, and for RQ3 we compare the differences between them.

**Regarding the Elicitation Approach, as Shown in Figure 4,** the elicitation with stakeholders was the most mentioned (63%), which is a positive point since user involvement is among the main success factors for projects [21]. The reference from other systems (39%), restrictions imposed by legacy systems (33%), and regulation restrictions (28%) were also mentioned with significant frequency. The second most cited approach was “based on professional experience” (47%), even if systematically, this approach may be less suitable for requirements such as response time and flexibility, as they would be based on experience, with possible incompatibility with the actual requirements of the system. Other answers, such as metrics collected in production (33%), problems found on the project (19%), and complaints from users (32%), reveal that the elicitation was performed in later stages.



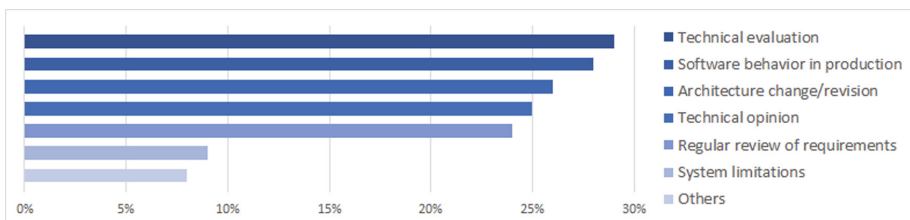
**Fig. 4.** NFRs elicitation approach

Figure 5 presents the answers during each activity the participants declared that the NFR was defined. As expected, in most cases, the requirement was elicited in the initial stages of design and analysis, however there are also answers reporting moments after the software delivery.



**Fig. 5.** Activities in which NFRs were defined

When asked if the requirement changed or was reviewed during the project, 25% answered that it was reevaluated but not modified and 32% that it was modified. Figure 6 presents the distribution of the answers about the drivers of change in NFRs, in the instances in which a review or change was reported in the previous question. In 87% of the cases, more than one reason was reported. With a higher percentage, there were options revealing that requirement changes were a reaction to something that happened in the system, such as changes in architecture, behavior or tests in production. The options “technical evaluation” and “regular review of requirements” represent more systematic approaches and received a significant number of answers (25% and 24%, respectively).



**Fig. 6.** Driver of changes in NFRs

The final question about NFRs was if the current requirement state is according to the system needs. To this question, 25% said that the requirement is outdated, and 9% did not know.

## 5 Discussion

### 5.1 RQ1 - How are Non-functional Requirements Defined in the Software Development Process?

Figure 5 presents the distribution of software development activities where the NFRs were defined. The results show that 22% were determined during development, and, for around 19%, their definition was reported to have happened in activities typical in later stages, such as based on user feedback, monitoring software in production, or testing. This result confirms that some requirements are neglected during the initial phases of the project, being only introduced when some event in the project points to that need. The literature reinforces the importance of this definition at the beginning of the project for technology selection and choice of patterns [25]. One participant reported for response time: *“the requirement was not idealized in the conception or design, a performance was expected, and during the tests we realized the need to include the requirement”*.

Some participants mentioned bad consequences of a late identification of requirements. For portability, one respondent mentioned that *“due to its late entry, it was adjusted and limited for having only a few capabilities”*. This limitation on implementing an NFR identified later was also reported regarding flexibility. The current literature supports this result [1,34], highlighting possible negative consequences of NFR late definition.

**Finding 1:** *A significant number, almost 1/3, of NFRs are not identified at the beginning of the project.*

The results show a great diversity of methods and techniques to elicit NFRs, often in combination. The fact that in almost half (47%) of the cases professional experience was reported as the approach used to define the requirement indicates the absence of a systematic approach for NFR elicitation. The number of answers that said the identification was based on problems found or user complaints also evidence that this later identification brought bad consequences to the project. For example, a respondent said *“problems that emerge only at deploy time on the client”*. The work of Lauesen [26] points out this lack of fulfillment of the requirements as a cause for project failures.

**Finding 2:** *Almost 1/3 of NFRs are defined in response to problems found on the project.*

This elicitation deficit is also reflected in the number of NFRs that changed later or are outdated to the system’s needs in production. According to the results obtained, 34% of the requirements reported are not up to date with the system in production, or its consistency is unknown. That is evidence that is common to have NFRs neglected until the production stage. Indeed, a recent study [37] pointed out not-clear and not-measurable NFRs as a problem in some agile processes.



This inconsistency with the system in production can also be explained by unnecessary NFRs included at the beginning of the project, which were ignored for not being important, but were not removed from the system requirements. One participant stated that sometimes that is noticed only *“when the technical proof of its infeasibility or that it is something unnecessary”*. In the same testimony, it was also mentioned that *“the costs presented to end users often make them rethink whether something is really necessary”*.

**Finding 3:** *More than 1/3 of NFRs are identified in the project are outdated or have an unknown state to the system in production.*

The answers about the qualitative versus quantitative approach for defining NFRs might indicate one of the reasons for this deficit, revealing that a more qualitative and non-objective way to define the requirements is frequently used. That was confirmed by a developer whose answer to the survey stated that the specification is deficient *“often not knowing how to elaborate the artifacts as well as the structure of the requirement text without objectivity”*. Another participant gave some examples of this lack of objectivity, mentioning *“good response time”*, and *“low cost for evolution”*.

## 5.2 RQ2 - How Non-functional Requirements Change and are Updated During a Software Project?

The responses to the survey revealed that around one-third of the NFRs changed during the project. Considering the approach used for the requirement elicitation described in the previous section, there are factors that could contribute to changes in the NFRs due to flaws in its specification, like the usage of non-systematic approaches and a qualitative and non-precise specification. Lack of prioritization of NFRs is associated with rework and customer complaints [34].

**Finding 4:** *The difficulty in dealing with the uncertainty and volatility of NFRs results in reactive actions.*

The reasons for changes or revisions of the NFRs are diverse, but there is a strong relationship between the changes and the absence of a deeper analysis of the real needs of the software. Absence or ineffective processes of continuous management of NFRs, and strategies to anticipate and accommodate possible changes [2,27] result in reactive actions and with impacts to the project [12]. Of the five most cited factors, only two result from a systematic assessment of requirements, while the other three result from the demand for changes in the architecture and behavior of the system in production. In addition to NFRs' late elicitation, these results about the motivation of changes also evidence a late validation of NFRs. Around 54% of the mentioned modifications occurred only after analyzing the behavior or tests of the system in production, which can add significant costs [12], and 35% required or were caused by architecture review or system limitation.

**Finding 5:** *A significant occurrence of changes on NFRs occurred after analyzing the behavior of the system in production.*

Several statements spontaneously written by the respondents point to the impact of the changes of NFRs on the solution’s design and architecture, even not having any question about it. As explicitly stated by one of them, changes in NFRs “*fundamentally change the design and architecture of a system*”. One respondent mentioned that the lack of visibility of the flexibility NFRs led to a revision of the architecture to support a higher number of customers. Another one mentioned that the system did not scale as it should, but it was in production without many complaints from customers. About system reliability, one mentioned that “*due to the addition of new features and new error situations, there came the need to change the requirements*”. Because of that, the architecture was modified, allowing more monitoring and failure recovery mechanisms. Since architectural decisions are usually made in the beginning of the project [40] and their implementation can be so interwoven in the code, changes might not be feasible [38].

In the other direction, changes in the architecture might also cause a review in NFRs. One respondent reported that after a change in the architecture “*portability requirements (servers, platforms and database) have been revised*”. The cost to implement the NFR was also mentioned by another participant as relevant to reconsider the requirement, revealing that some NFRs might be abandoned due to trade-off analysis that give them up in exchange of some other factor.

**Finding 6:** *Absence of a testing strategy makes it difficult to accommodate and validate changes that cross-impact NFRs and architecture.*

The close relationship between NFR and architecture results in managing and planning properly to avoid major impacts [24]. In this way, it is essential to pay attention to an adequate project testing strategy, anticipating and monitoring changes to mitigate them and be prepared to accommodate them [2].

### 5.3 RQ3 - How Much the Approach for Handling NFRs Vary Based on its Type?

For the previous research questions, we considered all the 144 reports from all six NFR types targeted in the survey. To answer this one, we analyzed the main differences between the answers given for each type. The first point to be highlighted is the number of participants reporting each requirement. As shown in Fig. 3, response time received the highest number of answers (75%), and the simultaneous number of requests received the lowest (51%).

Regarding elicitation approaches, the two most mentioned in all types of requirements were “elicitation with stakeholders” and “professional experience.” Scalability and portability did not have any other elicitation method to be highlighted, but they were the ones with the higher number of answers that reported

the elicitation with stakeholders, respectively 76% and 69%. For response time and amount of simultaneous requests, which are easier to measure and observe than the others, the reference from concurrent systems (respectively 54% and 47%) and measurements (respectively 46% and 58%) were mentioned more. Complaints from users received a significantly higher number of occurrences for response time (68%), which was more than double the overall percentage. Finally, regulations and standards were mentioned more frequently for flexibility (38%) and restrictions from legacy systems for robustness (50%).

The following activities can be highlighted as being higher than the overall percentage: conception for portability (36%), analysis for scalability (20%), design for flexibility (29%) and robustness (30%), and refinement for the amount of simultaneous requests (26%). For response time, no activity alone received a high number of answers, but it received most answers in activities more typical of late project stages, such as development, tests, and user validation.

**Finding 7:** *NFR identification might happen for different reasons depending on its type.*

Regarding requirements changes, the NFR type that received less reviews or changes was robustness (60% reporting no changes) while on the other extreme were portability (36%) and amount of simultaneous requests (30%). Portability was the most common to be reviewed but not changed (41%) and amount of simultaneous requests the most changed one (45%).

The NFR types most affected by changes or revisions in the architecture are scalability (45%), portability (44%) and flexibility (44%). While for response time and robustness, there is not a specific driver that stands out, for the number of simultaneous requests, the software behavior in production (41%) and experiments/evaluations made by the technical team (47%) were the main drivers.

**Finding 8:** *The most common drivers for change might be different for each kind of NFR.*

Although the survey revealed some common points to handling NFRs, each type of requirement has its particularities. Based on that, suitable techniques for elicitation and evaluating its suitability during the project should be employed based on the target requirement. This result is aligned with another study [34] that state that there no unique solution that will solve all requirements engineering needs.

## 5.4 Threats to Validity

Empirical studies face certain validity threats which can be of construct, internal validity, external validity and reliability [39]. Below, we describe the threats for this study according to this classification and how we mitigated them.

**Construct validity** refers to ensuring correct operational measures for the concepts (constructs) under study [39]. Since most of the questions regarded to the description of real experiences from the respondents, threats in this regard are limited. However, there is still a threat that respondents will not understand the responses in the same way as the authors. To mitigate this issue, we carried out a pilot to get feedback and perform some adjustments. Also regarding this threat, the survey ask participants to answer based on a specific project, instead of reporting a general experience.

**Internal validity** is associated with causal inference and happens when it is believed that a certain condition is a consequence from a second condition when, in reality, they are consequence of a third condition not considered in the study [39]. Since this study is exploratory and it is not claiming the existence of causal-effect relationships, this threat is reduced.

**External validity** is about generalizing the results to a large population [39]. As discussed in Sect. 3, this study has an exploratory goal and does not aim to have a generalizable description of how NFRs are handled. The goal was to show evidence of potential issues. Another point reinforcing external validity was the fact that the study was directed to professionals with a specific profile without previous definitions or suggestions, because the survey does not influence the project chosen by professionals at the time of the answer. Besides that, as mentioned earlier, this study is restricted to six types of NFRs. We acknowledge this limited scope however, by focusing on limited facets, we were able to collect more precise answers from the respondents. Consequently, even though the results should be understood as restricted to these types, they suggest the value of further investigation of this research problem.

**Reliability** is related to the dependence of the data and results on the researchers who performed the study [39]. To improve this aspect, we defined a protocol, including a questionnaire and data analysis procedures. We also made public the questionnaire to allow the replication of the study.

## 6 Conclusions

NFRs are essential aspects impacting the success of a software project. Although some research focused on them, there was a lack of evidence related to changes and their late definition. This paper presents a survey with experienced professionals on how NFRs were elicited and modified in concrete software projects to fulfill this gap. We found a lack of a well-defined step for evaluating and reviewing the NFRs, and a prevalence of changes in NFRs even in the late stages of development. This evidence highlights the need for agile techniques to deal with changes in NFRs, which can significantly impact the architecture if this uncertainty is not considered to make it ready to change.

The limitations of this study include investigations of specific NFRs, chosen to obtain more precise answers.

This limitation gives space for future studies directed to other NFRs. The information obtained in the survey helped to understand existing issues in the

management of NFRs in real projects, it was out of its scope to find the reasons behind these issues. Future works could also delve into the relationship between subjectivity in the definition of NFRs with the changes and real need for NFRs in projects and investigate whether the project management model influences the management quality of non-functional requirements during software development.

## References

1. Alsaqaf, W., Daneva, M., Wieringa, R.: Quality requirements in large-scale distributed agile projects – a systematic literature review. In: Grünbacher, P., Perini, A. (eds.) REFSQ 2017. LNCS, vol. 10153, pp. 219–234. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-54045-0\\_17](https://doi.org/10.1007/978-3-319-54045-0_17)
2. Ambler, S., Lines, M.: Introduction to Disciplined Agile Delivery - Second Edition. Project Management Institute, July 2020
3. Ambler, S.W.: Agile modeling. Wiley, Nashville (2002)
4. Ameller, D., Ayala, C., Cabot, J., Franch, X.: How do software architects consider non-functional requirements: an exploratory study. In: IEEE (2012)
5. Ameller, D., Franch, X., Cabot, J.: Dealing with non-functional requirements in model-driven development (2010)
6. Bajpai, V., Gorthi, R.P.: On non-functional requirements: a survey (2012)
7. Baltes, S., Ralph, P.: Sampling in software engineering research: a critical review and guidelines (2022)
8. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. SEI series in software engineering, Addison-Wesley Educational, Boston, MA (2003)
9. Berntsson-Svensson, R., Gorschek, T., Regnell, B.: Quality requirements in practice: an interview study in requirements engineering for embedded systems. In: REFSQ (2009)
10. Borg, A., Yong, A., Carlshamre, P., Sandahl, K.: The bad conscience of requirements engineering: an investigation in real-world treatment of non-functional requirements. In: Third Conference on Software Engineering Research and Practice in Sweden (SERPS 2003), Lund (2003)
11. Broy, M.: Rethinking functional requirements: A novel approach categorizing system and software requirements, pp. 155–187. John Wiley & Sons, Insc., September 2018
12. Budiardjo, E.K., Wibowo, W.C., et al.: Non-functional requirements (NFR) identification method using FR characters based on ISO/IEC 25023. Int. J. Adv. Comput. Sci. Appl. (2021)
13. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Springer, New York (2012). <https://doi.org/10.1007/978-1-4615-5269-7>
14. Dörr, J., Kerkow, D., Koenig, T., Olsson, T., Suzuki, T.: Non-functional requirements in industry - three case studies adopting an experience-based NFR method. In: 13th IEEE International Conference on Requirements Engineering (RE'05) (2005)
15. Eckhardt, J., Vogelsang, A., Fernández, D.M.: Are “non-functional” requirements really non-functional? In: ACM (2016)
16. Glinz, M.: On non-functional requirements. In: 15th IEEE International Requirements Engineering Conference. IEEE, October 2007

17. Guerra, E., Aniche, M.: Achieving quality on software design through test-driven development. Elsevier (2016)
18. Hoyer, R., Hoyer, B.: What is quality? *Quality Progress* (2001)
19. IEEE: Standard glossary of software engineering terminology. Std 610.12 (1990)
20. Jarzbowicz, A., Weichbroth, P.: A qualitative study on non-functional requirements in agile software development. *IEEE Access* **9**, 40458–40475 (2021)
21. Johnson, J., Mulder, H.: Factors of succes 2015 (2015)
22. Kirner, T.G., Davis, A.M.: Nonfunctional requirements of real-time systems. In: *Advances in Computers* (1996)
23. Kitchenham, B.A., Pfleeger, S.L.: Principles of survey research part 2. In: *ACM SIGSOFT Software Engineering Notes* (2002)
24. Knauss, E., Liebel, G., Schneider, K., Horkoff, J., Kasauli, R.: Quality requirements in agile as a knowledge management problem: more than just-in-time (2017)
25. Kumar, D., Kumar, A., Singh, L.: Non-functional requirements elicitation in agile base models. In: *Webology* (2022)
26. Lauesen, S.: IT project failures, causes and cures. In: *IEEE Access* (2020)
27. Leffingwell, D.: *SAFe 4.5 reference guide*. Addison-Wesley Educational, Boston, MA, 2 edn., July 2018
28. Mijanur Rahman, M., Ripon, S.: Elicitation and modeling non-functional requirements - A POS Case Study. *arXiv e-prints* (2014)
29. Nguyen, Q.L.: Non-functional requirements analysis modeling for software product lines. In: *IEEE* (2009)
30. Oriol, M., et al.: Data-driven and tool-supported elicitation of quality requirements in agile companies (2020)
31. Pohl, K., Rupp, C.: *Requirements engineering fundamentals*. Rocky Nook (2015)
32. Rahy, S., Bass, J.M.: Managing non-functional requirements in agile software development. *IET Softw.* **16**(1), 60–72 (2021)
33. Rainer, A., Wohlin, C.: Recruiting credible participants for field studies in software engineering research. *Inf. Softw. Technol.* **151**, 107002 (2022)
34. Sherif, E., Helmy, W., Galal-Edeen, G.H.: Managing non-functional requirements in agile software development. In: Gervasi, O., Murgante, B., Hendrix, E.M.T., Taniar, D., Apduhan, B.O. (eds.) *Computational Science and Its Applications – ICCSA 2022*. ICCSA 2022. LNCS, vol. 13376, pp 205–216. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-10450-3\\_16](https://doi.org/10.1007/978-3-031-10450-3_16)
35. Slinkas, J., Williams, L.A.: Automated extraction of non-functional requirements in available documentation. In: *1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)* (2013)
36. Svensson, R.B., et al.: Prioritization of quality requirements: State of practice in eleven companies. In: *IEEE* (2011)
37. Wagner, S., Fernández, D.M., Felderer, M., Kalinowski, M.: Requirements engineering practice and problems in agile projects: results from an survey (2016)
38. Waterman, M., Noble, J., Allan, G.: *How much up-front? a grounded theory of agile architecture* (2015)
39. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: *Planning*. In: *Experimentation in Software Engineering*, vol. 9783642290, pp. 89–116. Springer, Berlin, Heidelberg (2012)
40. Yang, C., Liang, P., Avgeriou, P.: A systematic study on the combination of software architecture and agile development. *J. Syst. Softw.* **11**, 157–184 (2016)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



# **Agile in the Large**





# Striving for Freedom in a Large-Scale Agile Environment with an Entrepreneurial Mindset of a Product Owner

Piret Niva<sup>(✉)</sup>, Maria Paasivaara, and Sami Hyrynsalmi

LUT University, Lahti, Finland

piret.niva@iki.fi, {maria.paasivaara,sami.hyrynsalmi}@lut.fi

**Abstract.** In a large-scale agile environment, a Product Owner receives requests from many different directions. Freedom to influence the direction of the product and push ideas forward sometimes requires saying “no”. This is a case study that has been made by interviewing several Product Owners or people working in a Product Owner type of role. The case company, which is a large financial organization, encourages Product Owners to take responsibility by valuing an entrepreneurial mindset. This research examines whether it is possible to exercise entrepreneurial freedom in the Product Owner’s work, and how much freedom the Product Owner has in the direction of the product, i.e. whether they have the freedom to say “no”. A total of 18 Product Owners, and those in similar roles, as well as managers from the case company, were interviewed. The findings show that the role of the Product Owner needs to be clarified in order to have more freedom to act. Prioritizing is difficult and saying “no” is more difficult than desired. Product Owners find the urge for an entrepreneurial attitude understandable, however, it does not seem to fit perfectly into the everyday work life of a Product Owner in a large-scale set-up. When the understanding of the role deepens, Product Owners could have greater freedom to make their products successful.

**Keywords:** Product Owner · freedom · large-scale · Agile · entrepreneurial mindset · prioritization

## 1 Introduction

The Product Owner role is a reasonably new role and comes from Agile software development. The name of the role refers to broad responsibility, for product ownership. According to the Scrum Guide [20] a Product Owner is accountable for maximizing the value of the product resulting from the work of the Scrum Team. In addition, a Product Owner is accountable for effective Product Backlog management [20]. Yet, there are undocumented requirements and organizational pressures which also impact the work of the Product Owner. In a large-scale Agile set-up the role needs to be scaled when several teams work for the same product and one Product Owner is not enough [16]. In a large-scale context also many

other roles and stakeholders exist and often bring requirements and pressure to the Product Owner. Thus, a question arises: How freely can Product Owners carry out their tasks and vision? Is the Product Owner role a fairytale?

The role of a Product Owner is more than taking care of the day-to-day running of the workday. According to Kelly [12], the role of the Product Owner brings little value if it is just backlog maintenance. The Product Owner should understand the technology and software development, as well as be able to say “no”, because the team cannot implement every idea [12].

Robert Frost was an American poet who once stated: “Freedom lies in being bold” [10]. Freedom speaks to philosophers, poets, and business creators. This study examines the freedom in the role of a Product Owner, especially how the Product Owner handles requirements and ideas coming from many directions in a large-scale Agile environment.

The Product Owner, who “owns” the product, can also be seen as a kind of entrepreneur. Our case organization hoped that Product Owners could lead their products with an entrepreneurial mindset. Therefore, this research explores how an entrepreneurial mindset is realized among Product Owners in the large-scale agile set-up of our case organization.

## 2 The Product Owner Role

The role of the Product Owner was introduced in the Scrum software development framework in 1995 [20]. Originally, Scrum was designed originally for one development team, with which one Product Owner would work. Nowadays, Agile is used also in large companies and projects. In large-scale agile, there are often many other stakeholders involved, and one product or service may have several Product Owners. Besides, working with the development team, the Product Owner may gather the requirements, communicate with the customers and even market the product, at least within the company, while the external marketing of a product is usually handled by marketing experts.

A symphony orchestra is a metaphor for how the Product Owner operates: a symphony is conducted by a conductor. Software developers can be thought of as symphony musicians. A symphony orchestra will certainly be able to play without a conductor. However, while musicians can play without leadership, they need to be in sync with others. The musicians make the choice to follow the conductor. The conductor helps to synchronize and makes the performance consistent. The conductor does not direct but orchestrates the symphony [22]. The Product Owner has a similar role as the symphony conductor: The Product Owner orchestrates the software development team. The developers (i.e. musicians), act independently, but the Product Owner takes the development forward in the direction which benefits the product (i.e. the music experience), which the whole team provides.

A Product Owner must have the mandate to play his or her role. A Product Owner collaborates within a network. Scrum does not place supervisory roles above the Product Owner. However, companies have their own internal organizational structures, and there may be more than one organizational model in

use. Kelly's [12] statement is a good starting point for considering how to own a product: "In the real world, managers report to owners. So surely Product Managers should report to Product Owners?"

The financial sector, where the case organization operates, is controlled, e.g., by regulations and social responsibility. Therefore its' operations are monitored by external agencies. The organization creates the boundaries for the Product Owner's role, and the level of freedom for the Product Owner, e.g., what the Product Owner can decide on his or her own, and when he or she can say "yes" to new requirements. For example, an entrepreneur is free to decide how to spend his or her time. This study investigates when the Product Owner feels free to say "no".

A Product Owner should not be a committee, she or he is one person [20]. Although, in the end, Product Owners are accountable, they may share the responsibility. For a Product Owner to succeed in work, an organization must respect the decisions made by the person in this role. The Product Owner role is also used by companies that may use a software development method other than Scrum, such as Kanban. Many of the Agile scaling frameworks, like the Scaled Agile Framework (SAFe) [19] and Large-Scale Scrum (LeSS) [21] have the Product Owner role. However, the role of a Product Owner is still largely implemented as Scrum states it.

Agility needs to be defined at an early stage in the organization so that all parties implementing it understand and are motivated by the goals. Developers' autonomy is one challenge as autonomy is difficult to maintain when the scale gets larger [4]. Software development is a people-to-people collaboration that seeks the same direction. This is one of the reasons why companies try new frameworks and customize them to their own needs. The Product Owner is not sitting in an ivory tower, but the role needs to collaborate [18]. An entrepreneur can make many decisions independently, however, the Product Owner is a role within the organization, and thus it does not have the same operating environment. The Product Owner acts as an interface between the stakeholders and the development team; the Product Owner represents stakeholders and communicates their needs in software development, and product creation [9]. Manifesto for Agile Software Development emphasizes responding to change, rather than complying with the plan [1]. Thus, the work of a Product Owner is expected to include elements that allow this role and the whole team to respond to changes.

Usually, Agile teams are small, five to nine-person teams [11]. When Agile is implemented in software development organizations with at least six teams or 50 or more people working on the same product, it can be called large-scale Agile [7]. This large-scale set-up causes complications, as often the basic Scrum set-up with one team working with one product guided by one Product Owner does not exist, but also the Product Owner role needs to be scaled and Product Owners need to collaborate inside the same product [16], which might impact on the experienced freedom and the entrepreneurial mindset of a Product Owner. The rest of the paper explores the experienced freedom and the entrepreneurial mindset of Product Owners in large-scale Agile.

### 3 Research Method

We chose case study [24] as the research method for this study, as it is suitable for exploring a new phenomenon. Interviewees from the case organization were chosen by purposeful sampling [17], which resulted in a diverse sample of Product Owners.

#### 3.1 Research Questions

As a Product Owner is accountable for effective Product Backlog management [20], one of the crucial aspects of the work is the prioritization of requirements. The first research question (RQ1) explores the freedom Product Owners experience in their work, and how they can control their operating environment by saying “no” while prioritizing requirements and giving direction for product development.

In our case company, there is a desire for the Product Owners to have an entrepreneurial mindset while striving for the success of their products. The second research question (RQ2) delves deep into whether the Product Owners feel that an entrepreneurial mindset is a good goal and how they themselves perceive their work from that viewpoint.

- **RQ1:** How does the Product Owner say no?
- **RQ2:** What is the entrepreneurial mindset of the Product Owner?

#### 3.2 Case Organisation

The case company is a Finnish financial company that has development units in a few large cities in Finland. 4500 ICT professionals work indirectly in the Development & Technologies organization of the case company, of which 1100 are directly employed. Some of the interviewees belong to the Business organization of the case company. In the past, the company’s software development worked largely according to the Scaled Agile Framework (SAFe) with a very large-scale Agile set-up. In early 2019, the entire company reshaped its organization to become more Agile by creating a new Agile culture. The case company sought to learn from Spotify’s tribal model and use it as a template. The financial sector has much stricter regulations than Spotify, an audio streaming provider, which has much more freedom to implement its platform. Thus, the case company modified its organization based on ideas from the Spotify model, and adopted, e.g., tribes.

In the case company, software development is mainly internal: development of new products, channels, and systems (e.g., applications, SaaS, interfaces, and core systems), or integration of purchased products. The developed systems are used for internal needs or by their customers. The portfolio is extensive, with accounts, payments, and insurance being its core business. The team size for the interviewed Product Owners in the case company was typically around 10 people, the smallest team size being four people and the largest thirty. Inter-team shared resources increased team size by an average of about three people. Usually, one Product Owner worked with one team. For those Product Owners, who had multiple teams, the total number of people in teams was several dozen.

**Table 1.** Interviewees

Interviewee ID	Educational background	Organization in case company	Length
H1	Business	Development and Technologies	41 min
H2	Technology	Development and Technologies	82 min
H3	Business	Development and Technologies	56 min
H4	Technology	Business	66 min
H5	Technology	Development and Technologies	78 min
H6	Business	Business	60 min
H7	Business	Development and Technologies	58 min
H8	Technology	Development and Technologies	62 min
H9	Business	Development and Technologies	42 min
H10	Business	Business	65 min
H11	Technology	Development and Technologies	62 min
H12	Technology	Business	76 min
H13	Technology	Development and Technologies	54 min
H14	Technology	Business	64 min
H15	Business	Business	59 min
H16	Business	Development and Technologies	53 min
H17	Business	Business	54 min
H18	Technology	Development and Technologies	71 min

### 3.3 Data Collection and Analysis

This case study explores a current phenomenon with empirical qualitative methods. Answering questions of *how* is a known strength of qualitative interviewing [2].

An advantage of interviews as a source of evidence is that they can focus directly on the research questions. Interviews are also useful for finding cause-and-effect relationships [23]. Semi-structured interviews were chosen as the main data collection method as it was desired to dive deep and explore a new topic.

All 18 interviews<sup>1</sup> were conducted in Finnish and transcribed, while citations were translated into English. The average interview duration was 61 min. The identifiers and the durations are reported in Table 1. Three interviews out of 18 were initial interviews, which were organized with managers and directors in Aug 2021. These interviews collected a deeper understanding of the case company and focused our research. The rest of the interviews with Product Owners and similar roles took place from September to October 2021.

Twelve of the interviewees were Product Owners and Senior Product Owners, two directors, one Expert Product Owner, one Business Developer, one Business Lead, and one Competence Lead. Senior Product Owners differ from

<sup>1</sup> The interview questions can be found at <https://doi.org/10.6084/m9.figshare.22087310.v1>.

Product Owners in the amount of experience. Expert Product Owners have even additional expertise according to the company's internal criteria, while Business Developers are entirely business oriented and often seen as equal to Product Owners. Of the 18 persons interviewed, half had technology as their educational background, while the remaining nine had business education. The interviewees were collected from two different internal organizations and mainly from two different tribes. We sought persons with solid expertise of product ownership, Agile or long experience in the case company. There were interviewees from several tribes, but some of the interviewees were sought from a certain tribe, because it was known that the tribe had expressed more challenges related to the Product Owner role and organization's practices for product ownership. Seven interviewees were from Tribe 1, four interviewees from Tribe 2, and seven interviews were conducted from case company's other tribes.

Traditional case study researchers have not sought a tabula rasa as a basis but have given room for analysis to evolve. A distinguishing feature has been the focus on the context and rich description of the phenomenon [8]. The ambition of this research was to allow room for an increase in perceptions of the role of Product Owners.

A benefit of online interviews is an efficient use of human resources, when participants are located geologically apart [14], as in this study. The interviews were conducted remotely as video meetings in Microsoft Teams. In three out of the 18 interviews, the call was conducted without a video image due to the wish of the interviewee.

The transcribed interviews were coded and themed<sup>2</sup> by the first author using a qualitative data analysis tool, Nvivo. With open coding, the codes emerged from the data and were grouped under higher-level themes. A total of 927 mentions were coded. For example, under the theme "entrepreneurship", with 59 mentions, were the codes: pros of entrepreneurship, cons of entrepreneurship, mindset, and rewards. The coded excerpts were carefully read and analyzed, and the main points were extracted as the results.

The results were validated by presenting them in a feedback meeting organized within the case company. An invitation to the feedback discussion was sent to over two hundred people, including all interviewees, and 28 persons participated.

## 4 Results

### 4.1 RQ1: How Does the Product Owner Say No?

In an entrepreneur's own company, the entrepreneur exercises supreme decision-making power, which offers freedom. This study deepens understanding of the concept of freedom by examining how a Product Owner can control the focus of product development by saying "no". The prioritized desires are usually new

---

<sup>2</sup> coding and thematization can be found at <https://doi.org/10.6084/m9.figshare.22434190>.

functionalities. When it comes to a large company that uses large-scale agile, like this case company, requests come to the Product Owner from many directions, and sometimes it is difficult for the Product Owner to say “no”.

**Production Problems:** There are situations concerning the operation and the future of the entire company, such as sudden production problems, which require quick prioritization of the repairing work. For example, a production problem that stops payment traffic is always prioritized above everything else. In these cases, the Product Owner serves the best possible way, and freedom is not exercised in any other way than prioritizing quick repairs.

*“If there is no production disruption... So to speak, to it [production problem] the Product Owner cannot say “no”. If it is in our context, the fact that services are up, so of course, that is always at the top of the list. But then to other parties or demands, the Product Owner can say “no” if there is a reason for it.”* — H4

**Company-Wide Objectives:** There are company-wide objectives with a lot of dependencies, over which the Product Owners do not have full influencing power. The highest objectives concerning the entire case company do not come as prioritized. Thus Product Owners feel that decisions trickle down too low in the organization: Sometimes the Agile model was felt unnecessarily decentralizing decision-making.

*“It is a bit like that, maybe the kind of Achilles heel of such an Agile, self-steering model, that it can kind of trickle down unnecessarily low the decisions about what to do and what not to do. Yes, there could be such optimization of resources at a higher level, and then giving more clearly that this is what we want.”* — H15

**To Whom Can the Product Owner Say “No”?** When the interviewed Product Owners were asked, who they can say “no” to, then seven out of 15 interviewees (excluding the first three initial interviews) replied that they can say “no” to anyone, but the answer came with laughter. The physical reaction was similar for each interviewee. 47% of interviewed Product Owners said, accompanied by laughter, that they can say “no” to anyone in the organization. However, the Product Owner’s own disbelief was manifested in the stir of laughter. Thus, they were not quite behind this opinion. They especially felt that when a superior, higher in the organization’s hierarchy, has greater decision-making power, it is difficult to give a negative or opposite opinion.

*“Not really to anyone. To the supervisor, you can’t really say “no”, it must be expressed differently in that situation.”* — H17

*“I tried to say “no” to my supervisor in some project, but the supervisor stated that it just has to be done. And then we did it [says the interviewee with a resentful sneer]. I think we would need to be able to say “no” if it is justified.”* — H6

Some of the interviewed Product Owners felt that they had a set of peers, e.g., other Product Owners, with whom they can have an equal say.

**Organizational Hierarchy:** The Product Owner has people above him or her in the organizational hierarchy. If the Product Owner wants to make changes, in the current model, the message must be taken up one step at a time. In this case, the next step in advancing a cause or a vision may be a business owner, followed by tribal leadership, one or two higher superiors, and finally the leader of the entire company. At every step, a new role takes the message forward. In this case, each step acts as an interpreter. The risk is that the perception of a product's needs or capabilities will shape along the way. Technical issues are interpreted in the business language in different ways by different people. Misunderstandings or message-modifying summaries are generated. Although the Product Owner takes the message up a certain hierarchy, the common supervisor of the people on these stairs, and the Product Owner, may be far away from each other in the organization's hierarchy. That is, not everyone above the Product Owner in the organization chart is superior to each other.

Saying “no” can be equally difficult for the superiors of the Product Owners. However, it does not help if the next step in the organization is facing the same dilemma.

*“Basically, the higher you go in an organization, the harder it gets, in a way you’ll have a sort of view, that you always have to respect and think about the position of the supervisor and his supervisor and his supervisor. But under certain conditions, you can say “no” to them. It is basically [a situation] where you can at least ask why you say that.”* — H15

*“In a way that... ”This kind of functionality has to be done right now”. It may be that the Product Owner sees that it has no value. But since the Product Owner can’t directly tell it to him or her, and then the person above the Product Owner doesn’t dare to say it up there, then it kind of doesn’t help anything.”* — H8

The companies have intentionally created hierarchies, but hierarchies should also be viewed in terms of how those could be enabling factors of product development.

**The Ways “No” is Said:** Figure 1 shows the ways in which the interviewees felt that the Product Owner could say “no”. Usually, the Product Owner handled the situations *through discussion*. To a certain point, the way to say “no” is to discuss. Communication skills are strongly emphasized in the work of the Product Owner. Although progress in the negotiation also depends on the goals of the participants of the discussion.

*“[When visions are at intersection] I feel that I have been able to communicate quite well the direction where we are going here [...] Sometimes there has been something smaller [requests], that has been said to be really important to get. Then I have pointed out that okay we will deliver, but please understand that this will then cause a delay in these other features.”* — H15

After discussion, the item might be *taken into the backlog and prioritized* according to its necessity or urgency. Some of the interviewees felt that they were under pressure from the business side, that the tasks *had to be taken into*



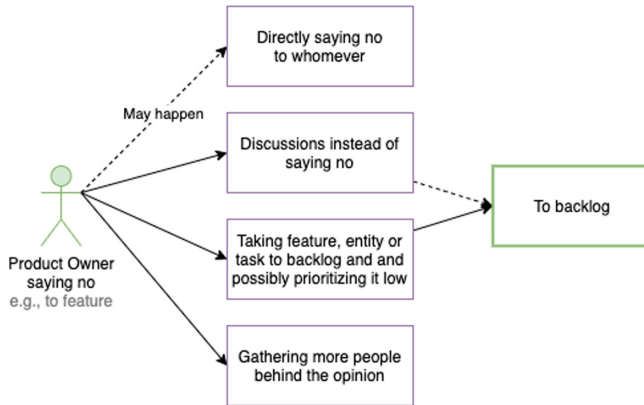


Fig. 1. Ways for the Product Owner to say no.

*the backlog*, even though the Product Owner did not think they were reasonable choices for the product as a whole, and thus might prioritize the items low. There is a risk that the item may be left in the backlog for a long time. While negotiating, a Product Owner sometimes has to say “yes” instead of “no”, so that the negotiation partner treats the ideas with an open mind and returns the favor by also saying “yes”. However, not all decisions can be traded.

*Gathering opinions behind* the Product Owner’s opinion is another way of saying “no”. The Product Owner gathers a few agreeing experts to a meeting to justify his or her own opinion. In this case, the Product Owner’s opinion was perceived to have more power, and a stigmatizing choice to oppose did not apply to the Product Owner anymore.

*“If you’re in a situation, where you can’t really get ahead between the two of you, then maybe you can take a few people or the issue owner, and look at the whole thing together. Then it will always be easier to make that decision and the big lines when more than one opinion agrees.”* — H4

**Dictating Culture and Not Understanding Agile:** Some interviewees felt that saying “no” is not constructive because the Product Owner does not have enough to say and the role is walked all over. In that case, the Product Owner would need help from the roles higher up in the hierarchy to say “no”, but these higher roles might not take a position on behalf of the Product Owner for their own reasons, e.g., because of conflicting goals or opinions.

*“We have such a culture, that top management can dictate. That “this is done like this” and no one can say anything about it. No one dares to say to top management. If we talk about that top management, the Product Owner is so far away from that, that he or she has no chance of saying anything up there. It would require that in a way our Business Leads and tribal leadership, would say “no”.”* — H8

*“If I would want to take a position on the orders which are coming from Business Leads and question that model, then yes, I feel like I can say “no”. But I know it is not far-reaching, because there are such strong personalities, that it comes back pretty quickly as a boomerang. It is not constructive, at least in that tribe. That kind of conversation cannot be discussed in a constructive spirit. Although I can say “no”, it doesn’t make sense in that environment.”* — H9

Some interviewees had unpleasant experiences of how opinions and know-how can be walked over. Shouting at meetings crosses boundaries that would not generally be wanted to be crossed. When the Product Owner’s role is not understood, some may abuse their power and customers might not get the best possible version of the product.

*“It should not be this HiPPO-style: that the Highest Paid Person decides. We make these products for customers. Through it, we reflect on those outcomes. But that’s it. This is often a matter of making compromises.”* — H11

Several Product Owners felt that there are situations when the business has not understood the Agile way of operating and walks all over the Product Owner.

*“If you say “no” too many times, and even if you justify it by the fact of moving to this Agile model, when you say enough “no”, you will be blacklisted. If you say once or two say so, you may not be blacklisted. It sometimes resembles some junior high school frankly. There, really, in some group meetings, the volumes may rise, and swear words are thrown. You don’t want to go along with that intentionally when you know what will follow. I prefer to swallow my disappointment and say that this is how it is done here.”* — H8

**The Product Owner Role Needs Clarification:** Several interviewees felt that Agile was not fully understood and the Agile roles would need clarification from a higher level of management. While transitioning to large-scale Agile the roles were defined in the internal intranet. However, this was felt insufficient. Especially the Product Owner role would need clarification: how the role is seen, how it is positioned, and what kind of power and freedom the Product Owner has. If the Product Owner is to lead the product, then he or she should be able to make decisions and show the direction.

*“There must also be decision-making ability and not everything can be discussed in the Swedish style until the end or endlessly. But it is good to go through things. Then, if there are differences of opinion on which direction to take, then, of course, the role of the Product Owner is to make decisions and, in a way, show direction.”* — H4

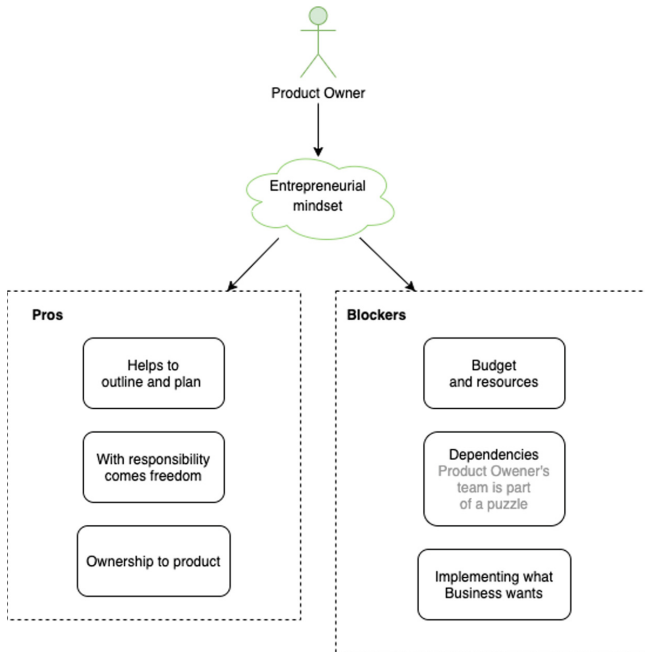
The interviewees felt that it is difficult to be responsible for the success of a product if its direction is not determined by the person responsible for it. It emerged also from the director-level interviews that, the Product Owner is expected to resist by saying “no”, to hold the side of the product and that way make it better, but in practice that turned out to be difficult. Finally, a clarification was wanted on how IT and business should work together, in an Agile way.

## 4.2 RQ2: What is the Entrepreneurial Mindset of the Product Owner?

The case company emphasized how an entrepreneurial mindset could benefit the work of Product Owners. Any small and large company can create its own products. An entrepreneur “owns” his or her own company’s products and takes them forward so that it will be a reasonable business, justified by its technological solutions and created customer value.

The decision-makers of the case company expressed in the interviews that they wanted to encourage the Product Owners to have an entrepreneurial mindset. They hoped that the Product Owners would take ownership of the product, that the entrepreneurial mindset would help them to outline and plan the product, and finally, by taking responsibility the Product Owners would have also more freedom. This mindset was expected to benefit the company.

Our interviews revealed that there were still many blockers on the way and room for improvement in the support structures enabling the entrepreneurial mindset. The case company could benefit from giving more opportunities and freedom so that the Product Owners would get to show their full capability and skills. The aspects of an entrepreneurial mindset in the case company are described in Fig. 2.



**Fig. 2.** Pros and blockers of an entrepreneurial mindset.

**Attitude Towards the Entrepreneurial Mindset:** The interviewed Product Owners were motivated and committed to their work and felt that they could use their skills in this role. 39% of the interviewees raised positive issues about the entrepreneurial mindset, while 56% raised negative issues.

Entrepreneurship was felt to be largely about being able to build something new from scratch.

*“As an entrepreneur, if and when you get to build a completely new service from scratch, I think that is more entrepreneurial”* — H6

Even when the interviewees felt that the case company had not given Product Owners an opportunity to be entrepreneurial, an entrepreneurial mindset was perceived as a good goal within the company.

*“I don’t know how to rationalize [entrepreneurship as a goal], but yes you have to be truly independent and forward-looking in the best possible way relative to your own product or area.”* — H7

*“Absolutely! And this [entrepreneurial attitude] is one element that I personally love above everything else.”* — H1

*“I see that those are my stuff, I own those, and I have to take that forward. And that’s clear. But I think it is a pretty natural pattern of thought. That’s how it should be for everyone, though. [...] When it comes to entrepreneurship, I might feel it is a little different than what it really is in a bigger company like this. It is more of a sphere of thought that you need to be entrepreneurial here maybe, but the activities might be a little different nonetheless.”* — H13

Regarding the Product Owner’s motivation and goals, an interviewee presented an analogy of a sports hobby. An entrepreneurial mindset can mean the same as taking responsibility in a sports team or in a goal-oriented sport. The athlete is always able to invest in his or her own development with independent decisions.

*“I find the analogy more from the world of team sports, than from entrepreneurship. And yet the goal in both is the same, that we are just super committed and striving to achieve the goals.”* — H18

**Challenges Regarding the Entrepreneurial Mindset:** Although the requirements of an entrepreneurial mindset were understood, our interviewees reported many factors why a Product Owner could not be entrepreneurial in our case company: 1) The Product Owners could not create their own vision for the product but had to implement what the business wanted. 2) In a large-scale Agile setup there were a lot of dependencies across the organization, thus decisions could not be independent. 3) The budget and resources were decided elsewhere in the organization and the Product Owners did not even have much to say on how to use the budget. Next, each of these challenges is discussed.

**Implementing What Business Wants:** An entrepreneur creates a *vision* for his or her product, which the interviewed Product Owners felt they did not have

a possibility to do. Instead, new features came from elsewhere, from the business side, thus it was felt that the Product Owner was losing the opportunity for an entrepreneurial mindset. Instead, the Product Owner was just implementing what the business wanted. Whereas an entrepreneur would own the product and could decide the direction.

*“I feel like I work as an IT team coordinator and a responsible person, but not as an entrepreneur. I feel that an entrepreneur should be creating that vision, and there should be possibilities to make an impact vertically, not just horizontally. It [the work of Product Owners] completely lacks the aspect of effect vertically. That’s why I don’t think there’s anything left of entrepreneurship in it, just crumbs.”*

— H9

*“When it comes to just queuing up what comes from business as an input, that is not entrepreneurial. Or I don’t know, ditch excavation can also be entrepreneurial, you can start a business for that too.”*

— H2

If the Product Owner is seen more as a performer of tasks than as an orchestrator of the whole, then product ownership and thus entrepreneurship will not materialize.

**Dependencies:** In the large-scale setup of the case company the team of each Product Owner was seen as just part of a puzzle - it had a lot of dependencies on other teams and other parts of the organization. Thus, the Product Owner did not have the possibility to make decisions independently.

*“But on the other hand, then, you can’t just be an entrepreneur [in this role] who makes decisions completely independently, but there are a lot of dependencies, which, in a way, then drops away the other side of the entrepreneurship, that is, the maneuver space.”*

— H14

In addition, the Product Owner was not responsible for the whole, but just part of the work done for the product. For example, the maintenance did not belong to the Product Owner’s area. Thus, the current model was not seen as supporting the entrepreneurial mindset.

*“Everyone probably wants to act like that [with an entrepreneurial mindset], and everyone wants to look like it works. But an entrepreneurial attitude means that you really own your product and are serious about getting that better and more profitable. We have thrown out maintenance from products elsewhere. And then things related to the user experience are in pretty bad shape overall. And then we are required to do 120% of the work time new features, so try to be entrepreneurial there! [...] It may be that people are like fooling themselves with it, but in this model where we are now, it does not work. We don’t really have product ownership, we don’t have product management. Then when we have them, then we can be entrepreneurial.”*

— H2

One of the dependencies mentioned was house-level policies posed for all employees. As an example, an interviewee gave the remote work policy. It was hoped that Product Owners would be able to influence the amount of remote

work they do. It would be difficult to see the environment as entrepreneurial if the employer dictates how much the employee or team is allowed to work remotely.

*“But then there are still certain house-level policies, in terms of practices or others, so that for example, I am looking forward with great interest to how strict the policies will be for us to come back to the office [after the pandemic]. [...] When you have the freedom to choose where you work, then you will have a certain responsibility to do the work. Because then there’s kind of no one watching all the time. In my opinion, entrepreneurship is based on that. That you have that freedom, but then you also have that responsibility.”* — H8

**Budget and Resources Coming Elsewhere:** It was felt that when the budget and resources come from elsewhere, it is difficult for the Product Owner to be entrepreneurial. Although an entrepreneur may not have even the same budget available as a Product Owner in a large-scale Agile company, at least entrepreneurs have more freedom to decide how to use the budget.

*“For that operation to be entrepreneurial here [in my team], in a way, this little team would be my own boutique... We are not there. Yes, it would take quite a few things. One would be the budget. Also, I also really should have an influence on things like resourcing.”* — H10

**Monthly Salary Creates Safety:** Several of the interviewed Product Owners from the case company felt that they are employees in a large-scale Agile environment with a monthly salary, which creates job security that they liked.

*“And in this context, that work is done for the employer.”* — H4

Making the environment more entrepreneurial would run the risk that employees would no longer feel so safe, and the focus would be on the potential negative aspects of entrepreneurship, such as job and salary insecurity.

*“I’ve always thought I wouldn’t want to be an entrepreneur, in a way, that safe job is a pretty important thing to me.”* — H17

*“Entrepreneurship means that you are a 24/7 entrepreneur, and, in a way, the workday does not end. Even though I personally feel that my workday does not end when I end the day, I’m still available after that if needed. But it’s kind of like I’m not responsible for being available 24/7.”* — H18

Thus secure job was seen as a positive aspect of the current situation in the case company.

### 4.3 Validation

A feedback meeting was held at the case company to validate the results of the analysis. The participants shared their opinions on the results, i.e. their views on the validity of the results. Eight participants (commenter IDs V1-V8) gave comments: they recognized the areas for development and the problems. The only concern raised in the discussion was that if the tribes of the interviewees are very different from each other, then the results of the study might be bundled together into a too-generic outcome. However, the purpose of this study was to obtain versatile data from different parts of the company. The results that this research underlined were perceived as identifiable.

*“It was very delightful to see that the experiences I have myself were reflected in this presentation. I think you have at least got into this presentation very well, how this world looks like from my point of view. [...] That is, there has been a slight debate in our tribe about who makes those decisions. One view there is that business tells us what to do, and the job of Development and Technologies is to do as we are told or do, i.e., what the business says.”* — V2

The commenters felt that there were still difficulties in implementing an Agile culture. For example, features for products were ordered, rather than discussed and decided together.

*“I feel like we are still pretty strong in there, what was being said, that features are being ordered.”* — V4

When considering the freedom of the Product Owner, the interviewees felt that the current resourcing and budgeting model does not quite support Product Owners to try out new ideas. Instead, resources are devoted to what is planned from the business side.

## 5 Discussion and Conclusions

Central to the development of modern capitalism has been the idea of freedom [6]. In this paper, we studied the freedom of the Product Owner role in a large-scale Agile organization regarding two aspects: how a Product Owner can influence the direction of the product by saying “no”, and whether Product Owners have an entrepreneurial mindset. To our knowledge, this is the first study exploring these topics.

An entrepreneurial mindset is perceived as important in engineering, and scholars emphasize the importance of creativity in science-oriented careers [5]. Prior knowledge is important for an entrepreneur to recognize an opportunity. An entrepreneur is able to create a venture when an opportunity has been identified [13]. The Product Owner creates products. An entrepreneurial mindset can be thought of as proactive and saying “no” as reactive. In this study, we found that the organization’s hierarchy and budget create boundaries within which

the Product Owner can say “yes”. However, the Product Owner needs to set limits and make room in the backlog for new opportunities by saying “no”.

**How Does the Product Owner Say No?** The interviewed Product Owners claimed they can say “no” to anyone, but the results showed the uncertainty behind this claim: In reality, they were not able to freely scope their products in this large-scale Agile setup. Instead, their superiors and business people could walk all over the Product Owners’ vision. Results revealed that Product Owners preferred handling the conflicting scoping decisions through discussion, instead of directly saying “no”. This could result in taking items to the backlog against the Product Owner’s vision, and in that case, aiming to prioritize those items low. Gathering support from persons having the same scoping opinion was another option to keep unwanted items out of the product scope.

The role of the Product Owner and its’ decision-making power was unclear in this large-scale Agile setup and would need clarification. Many felt that the organization lacked a proper understanding of Agile. The findings showed that the Product Owner role in a large-scale Agile organization may not be realized according to the Scrum guide [20].

**What is the Entrepreneurial Mindset of the Product Owner?** The underlying human predisposition of entrepreneurship to try or pursue persistently entrepreneurship is determined by different models [3]. Giving freedom improves organizational agility, and freedom empowers employees to act independently and in a self-coordinated way, as entrepreneurs do [15]. Thus, our case organization encouraged the Product Owners to take responsibility by valuing an entrepreneurial mindset. The interviewed Product Owners found this goal understandable, however, they reported many blockers that prevented reaching the mindset: 1) The Product Owners could not create their own vision for the product but had to implement what the business wanted. 2) In a large-scale Agile setup there were a lot of dependencies across the organization, thus decisions could not be independent. 3) The budget and resources were decided elsewhere in the organization and the Product Owners did not even have much to say on how to use the budget.

Product Owners are expected to lead the product to success. This study shows that the Product Owners perceive the entrepreneurial mindset to some level as a utopia in a large-scale Agile setup. It was felt to be more important that the Product Owner could make independent decisions.

**Limitations** of this study include the following: We interviewed 18 persons from a large-scale organization, thus we could triangulate the answers among the interviewees. However, we could not cover all Product Owners in the company, thus, some viewpoints might be missing. This is a single case study. To increase generalizability to other similar organizations, we aimed to provide a rich context description. However, studying several case organizations would have provided better possibilities to generalize.

**Future research** could dig deeper into how the Product Owner’s freedom could be concretized in a large-scale Agile environment. We encourage researchers to



perform similar case studies in other large-scale organizations to study how the freedom of the Product Owners is realized and what the implications are.

## References

1. Beck, K., et al.: Manifesto for Agile Software Development (2001). <https://agilemanifesto.org/>
2. Brinkmann, S.: *Qualitative Interviewing: Understanding Qualitative Research*. Oxford University Press, Oxford (2013)
3. Burke, A.E., FitzRoy, F.R., Nolan, M.A.: What makes a die-hard entrepreneur? Beyond the ‘employee or entrepreneur’ dichotomy. *Small Bus. Econ.* **31**(2), 93 (2008). <https://doi.org/10.1007/s11187-007-9086-6>
4. Conboy, K., Carroll, N.: Implementing large-scale agile frameworks: challenges and recommendations. *IEEE Softw.* **36**(2), 44–50 (2019)
5. Daspit, J.J., Fox, C.J., Findley, S.K.: Entrepreneurial mindset: an integrated definition, a review of current insights, and directions for future research. *J. Small Bus. Manag.* **61**(1), 12–44 (2023). <https://doi.org/10.1080/00472778.2021.1907583>
6. Dierksmeier, C., Pirson, M.: The modern corporation and the idea of freedom. *Philos. Manag.* **9**, 5–25 (2010). <https://doi.org/10.5840/pom2010932>
7. Dikert, K., Paasivaara, M., Lassenius, C.: Challenges and success factors for large-scale agile transformations: a systematic literature review. *J. Syst. Softw.* **119**, 87–108 (2016)
8. Dyer Jr, W.G., Wilkins, A.L.: Better stories, not better constructs, to generate better theory: a rejoinder to Eisenhardt. *Acad. Manag. Rev.* **16**(3), 613–619 (1991). <https://doi.org/10.5465/AMR.1991.4279492>. <https://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=4279492&site=ehost-live>. Publisher: Academy of Management
9. Felderer, M., Méndez Fernández, D., Turhan, B., Kalinowski, M., Sarro, F., Winkler, D. (eds.): *PROFES 2017. LNCS, vol. 10611*. Springer, Cham (2017). <https://doi.org/10.1007/978-3-319-69926-4>
10. Frost, R.: *Television: Men of Faith* by Philip Hamburger. *The New Yorker*, pp. 167–169, December 1952
11. Jyothi, V.E., Rao, K.N.: Effective implementation of agile practices. *Int. J. Adv. Comput. Sci. Appl.* **2**(3) (2011)
12. Kelly, A.: *The Art of Agile Product Ownership*. Springer, Heidelberg (2019). <https://doi.org/10.1007/978-1-4842-5168-3>
13. Kuratko, D.F., Fisher, G., Audretsch, D.B.: Unraveling the entrepreneurial mindset. *Small Bus. Econ.* **57**(4), 1681–1691 (2020). <https://doi.org/10.1007/s11187-020-00372-6>
14. Musselwhite, K., Cuff, L., McGregor, L., King, K.M.: The telephone interview is an effective method of data collection in clinical nursing research: a discussion paper. *Int. J. Nurs. Stud.* **44**(6), 1064–1070 (2007). <https://doi.org/10.1016/j.ijnurstu.2006.05.014>. <https://linkinghub.elsevier.com/retrieve/pii/S0020748906001672>
15. Nobles, B., Staley, P.: *Freedom-based management*. MLAB Labnotes (2009). <http://www.42projects.org/docs/FreedomBasedManagement.pdf>
16. Paasivaara, M., Heikkilä, V.T., Lassenius, C.: Experiences in scaling the product owner role in large-scale globally distributed scrum. In: *2012 IEEE Seventh International Conference on Global Software Engineering*, pp. 174–178. IEEE (2012)

17. Patton, M.Q.: *Qualitative Research & Evaluation Methods: Integrating Theory and Practice*. Sage Publications, Thousand Oaks (2014)
18. Saddington, P.: Scaling agile product ownership through team alignment and optimization: a story of epic proportions. In: 2012 Agile Conference, pp. 123–130. IEEE (2012)
19. Scaled Agile Inc.: Homepage: Scaled Agile Framework (2021). <https://www.scaledagileframework.com/>
20. Schwaber, K., Sutherland, J.: *The Scrum Guide* (2020). <https://scrumguides.org/scrum-guide.html>
21. The LeSS Company B.V.: Area Product Owner. <https://less.works/less/less-huge/area-product-owner>
22. Tiwana, A.: *Platform Ecosystems: Aligning Architecture, Governance, and Strategy*. Newnes (2013)
23. Yin, R.: A review of case study research: design and methods. In: *Applied Social Research Methods*, vol. 5, p. 219. SAGE Publications Ltd., January 2003. Journal: *Applied Social Research Methods*
24. Yin, R.K.: *Case Study Research and Applications: Design and Methods*, 6th edn. SAGE Publications, Los Angeles (2018)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# Sustaining Agility: Organizational Change, Factors and Theoretical Lenses

Leonor Barroca<sup>1</sup> (✉), Helen Sharp<sup>1</sup>, Advait Deshpande<sup>1</sup>, Peggy Gregory<sup>2</sup>,  
and Stavros Papadeas<sup>3</sup>

<sup>1</sup> The Open University, Walton Hall, Milton Keynes MK7 6AA, UK  
leonor.barroca@open.ac.uk

<sup>2</sup> University of Central Lancashire, Preston, UK

<sup>3</sup> The Business Agility Institute, New York, USA

**Abstract.** Agile organizations have to deal regularly with change and at the same time adapt to sustain agility. In this paper, we present an initial study to identify factors considered when changes need to be made to sustain agility. We used a novel data collection approach, critical decision method (CDM), and investigated three theoretical lenses, paradox theory, situation awareness and shared mental models, to explore the kind of practical consequences they help to uncover. This paper presents the findings of this initial study together with reflections on the data collection method and the three theoretical lenses. Three key dimensions relevant to sustaining agility emerge from the use of these theoretical lenses: teams vs organization; understanding the environment vs the impact of change internally; and understanding “now” vs looking into the future.

**Keywords:** Sustaining · agility · change

## 1 Introduction

Agile methods have been practiced for many years, and the challenges agile practitioners face have evolved over time. Initial concerns focused on how to adopt agile software development, and later, on how agile can be scaled to large IT projects [1]. More recently, challenges have moved towards business agility transformation [2], and how to remain agile in the long term [3, 4]. This paper focuses on this last concern, which we refer to as “sustaining agile”, i.e. the continuous process of maintaining and improving agility within an organization.

An organization that has transformed to agility, or in which only some part(s) of it have adopted agile practices may face different issues around sustaining agile [2] than one that has been agile from its inception. An organization that has adopted agile working from its inception may find the question of how to sustain agile puzzling because a key characteristic of agility is continuous adaptation and improvement, and one of the agile principles refers to sustainable pace: “The sponsors, developers, and users should be able to maintain a constant pace indefinitely”. However, by sustaining agile we are not referring to the continuous ongoing flexible adaptation of agile work but to how agile organizations deal with potentially disruptive change, yet continue to be agile.

© The Author(s) 2023

C. J. Stettina et al. (Eds.): XP 2023, LNBIP 475, pp. 115–131, 2023.

[https://doi.org/10.1007/978-3-031-33976-9\\_8](https://doi.org/10.1007/978-3-031-33976-9_8)

Studies on organizational change tend to be long term, involving many participants and huge amounts of data collection and analysis. On the other hand, short interview studies can be limited in their depth of insight. To overcome these extremes, we set out to explore a different data collection approach, and to investigate the theoretical frameworks that might shed light on the question of how organizations sustain agility through disruptive change. We also wanted to identify concrete factors, specific examples and practical recommendations that practitioners demand [5]. To that end, we have conducted an initial study to identify factors considered when an organization needed to put effort into sustaining agility. We used a novel data collection approach, critical decision method (CDM) [6], and investigated three theoretical lenses, paradox theory [7], situation awareness [8] and shared mental models, [9] to explore the kind of practical consequences they help to uncover. This paper presents the findings of this initial study together with reflections on the data collection method and the three theoretical lenses; it contributes to the research on sustaining agile by identifying an initial set of factors considered when changes need to be made, and by proposing a data collection approach that focuses on real practice and is more targeted than the typical longitudinal studies.

The next section surveys literature in sustainability, resilience and organizational change. Section 3 describes the study design and Sect. 4 presents its results. Three theoretical lenses and their ability to generate concrete insights are considered in Sect. 5. Section 6 presents lessons learned from this initial study and concludes the paper.

## 2 Related Literature

The term ‘sustainability’ is used in a wide range of contexts. It is nowadays commonly associated with the UN’s sustainable development goals [10] where it is defined as “meeting the needs of the present without compromising the ability of future generations to meet their own needs”. However, a more general definition of sustainability is “the ability to be maintained at a certain rate or level” [11]. This more general definition is reflected in the literature on organizational change [12–15] and in software engineering literature focused on sustaining agile [16–18].

Buchanan et al. [12] reviewed studies on sustaining organizational change and defined it as “the process through which new working methods, performance goals and improvement trajectories are maintained for a period appropriate to a given context.” They proposed a provisional model for the processes influencing sustainability but the studies they reviewed focus on financial, political, and contextual factors affecting change rather than agile concerns such as customer focus, adaptability to uncertain environments and team empowerment. Holbeche [13], on the other hand, focuses on organizational agility. For this author, organizational agility is the “capacity to respond, adapt quickly and thrive in the changing environment”, and a resiliently agile organization has:

- “an organizational culture and structure that facilitates change within the context of the situation that it faces;
- staff who are willing and able to give of their best – in a sustainable way; and

- a learning mindset in the mainstream business and underlying lean and agile processes and routines to drive innovation.”

This view resonates strongly with our notion of “sustaining agile”. In another work, Holbeche [14, p.54] lists the activities needed in the quest for business agility, gathering them into a circular model with 4 quadrants: agile strategizing, operations, linkages and people practices; at the centre and underlying everything are agile people and culture. But both [13] and [14] fail to address how organizational agility may be sustained. More recently, Miceli et al. [15] propose that sustainability and resilience are distinct but interdependent and that agility supports an organization in building resilience. However, they observe that research has still to identify the factors that are considered in the process of maintaining an organization’s agility.

So although there is a growing interest in agility within the organizational change literature, we found no studies that investigate how organizations sustain their agility.

There have been some investigations into sustaining agile within software engineering literature, but they are limited. Senapathi and Srinivasan [16] view sustaining agile with regards to the six stages presented in Rogers’ diffusion of innovation theory [19]. They see sustainability from a change management point of view and consider it as the post-adoptive stages where a change is accepted, routinised and infused into an organization. In this case, agile is being used, it becomes a normal activity, and it penetrates deeply and widely in the organization. Sedano et al. [17] investigate sustainability on the agile development team level. They define sustainable software development as the “ability and propensity of a software development team to mitigate the negative effects of major disruptions, especially team churn, on its productivity and effectiveness”. Barroca et al. [18], report on what practitioners thought agile sustainability meant. Four themes emerged from this study: being completely agile (whole organisation, mindset, and principles); being independent (learning and self-sufficiency); being focused on business value and need (user/customer need, value, business need, and appropriate use); and being consistent across time (sustainable pace, and leadership and appropriate technical skills).

Today this is still an under-researched area with little in-depth study of agile sustainability. We therefore formulated our research question as: ‘*What factors do IT organizations consider when making changes to sustain agile?*’.

### 3 Study Design

The overall framework chosen for this initial study is the critical decision method (CDM) [20]. This approach uses multi-pass retrospection in which the same event is described more than once, but with differing levels of detail. It is an extension to the critical incident technique [21] to investigate how decisions are made in real situations. It has been used to research team decision-making [8], emergency ambulance control [22], and software flaws [23]; it is an interview method aimed at eliciting specific concrete experiences rather than idealized accounts. CDM’s anchoring in concrete experiences makes it appropriate for our focus on specific factors and practical insights; it allowed us to have a more targeted focus in data collection than that provided by longitudinal

studies, while also providing the specificity that it can be hard to get from standard interview studies.

Participants were asked to identify “examples of particularly significant changes they needed to make in order to sustain agility”. More specifically, they identified a change that they believed was relevant to agile sustainability, described it and explained how it was effected. The CDM technique has not, to our knowledge, been applied to this kind of situation before. The study gained ethical approval from the relevant committee of one of our universities.

### 3.1 The Case Study: AgileCo

We sought a software development company that had experience of needing to make changes to their business in order to preserve their agility, and we were introduced to AgileCo through our collaboration with the Agile Business Consortium and the Business Agility Institute. AgileCo is a large video game developer with players all over the world. The company has been player-focused since their formation, and it is a desire to remain customer-centered that drives them, rather than an explicit need to sustain agility. But “*to be customer-centric we have to be agile*” (CTO). At the time of the study the organization employed around 3000 people and had 20 offices around the world. Most of their developers are also gamers themselves. Development takes place through iterations, with releases every 2 weeks. Each team is empowered to choose whichever approach they prefer (Kanban, Scrum, etc.), led by a delivery leader who is in charge of delivering the product and is also an agile expert supporting the team to work in the best agile way for them. This autonomy extended to how best to form teams, when to integrate teams, split teams or form new ones. This flexibility and the delivery-led structure extended to other parts of the business such as finance and HR where delivery leaders were also deployed. The company has a strong focus on education and all employees are taught the basics of agility from the moment they join the company.

AgileCo formed in the early 2010s. In the first 10 years they developed and maintained only one game and then the business decided to diversify and develop multiple games. The move to multiple games demanded the centralization of functions common to the different games, such as registration, but key decisions such as release cycles and feature development remained with the teams. This led to a situation in which development activities had to be co-ordinated more closely; teams were encouraged to release their games when ready rather than staggering them. This had implications for co-ordination and prioritization of centralized development. Moving to a multiple-game organization was a strategic decision taken in response to the changing external environment and was intended to maintain business value for the organization and its users, the players. At this time, the organization had highly distributed and empowered teams that knew, and had total ownership of, what they had to do.

This is the context within which we interviewed our participants.

### 3.2 Data Gathering

AgileCo were approached initially by our collaborators. Following a virtual introduction we interviewed the company's CTO who put us in touch with further interviewees who were more closely involved in relevant business changes.

The CTO interview elicited the company's background and was used to produce the description above. Our second interviewee had overall responsibility for two relevant business changes. Subsequently, our second interviewee recommended our third and fourth interviewees. Overall, we performed six interviews, four of which followed the CDM structure described above. The questions used during the CDM cycles are in Table 1. Note that the participants identified what constituted business change and that none of the questions ask directly about the factors considered during this experience. The second interview uncovered two changes, and the subsequent four interviews sought further detail and different perspectives on these two changes. Interviews were conducted and recorded through Microsoft Teams, and then transcribed.

**Table 1.** Prompting questions asked during the CDM cycles.

Topic	Question
Cues & Knowledge	Why did you choose this example to share with us?
Analogues	Is this a typical example?
Goals	What were you trying to achieve at the time?
Options	Did you try different resolutions (or is this a linear process)?
Basis of Choice	How was this resolution selected/other resolutions rejected? Were any rules or heuristics being followed?
Experience	What training, knowledge, or information did you draw on?
Decision-Making	Were you under any time pressure? How long did it take?
Aiding	Did you seek any help?
Reflection/Impact Today	What impact did this resolution have? If a key feature of the situation had been different, would it have made a difference to your decision?

### 3.3 Data Analysis

Data gathering resulted in six hours of transcribed interview data. Analysis was based on Wong's [24] CDM analysis guidance, specifically using the thematic approach. First, descriptions of the two changes identified by interviewees were extracted. Then the specific examples of these changes were examined. Using a reflexive thematic analysis as described by Braun and Clarke [25], factors that the interviewees regarded as important in effecting the business changes were identified. Throughout, our aim was to characterize the situation from their point of view.

Three researchers were involved in the analysis, independently identifying themes and then discussing them. The findings were reported back to the organization for member checking and it was agreed that they reflected their experiences.

## 4 Study Results

The two changes identified were both in the context of changing from being a single-game company to a multiple-game company: changing the work system to balance individual team and central organization concerns; and prioritizing items proposed by senior managers of the organization. The first emphasizes agility closer to the coal face (the middle/lower layers of activity), while the second emphasizes agility at the higher levels of strategic value.

### 4.1 Changing the Work System

Changes to the work system were triggered from within the organization itself because of the need for centralization of some functions. The diversity of tools being used by the teams (spreadsheets, Jira, Favro, Kanban) made it difficult for progress towards milestones to be visualised. So, they decided to introduce a new work system that allowed work to be tracked centrally, and be a single point for measuring progress. The process to define this new system was slow, and was carried out in collaboration with the teams, bringing them along and avoiding creating overheads to their work.

The challenge to sustaining agility in making this change is summarized as: *“not disrupting things at the coal face but still reap[ing] some benefits of predictability at the top”* and the main aim was *“finding the balance between empowering their [the teams’] agility and their best practices and giving the <...> large organization value”*.

To make this change successfully the organization had to find *“the pivot from individual teams owning and doing whatever they want into more of a web of collaboration between teams and discovering what that means and not having to argue from first principles that centralization is evil or wrongheaded every time an optimization is tried to be made for efficiency”*. As an agile organization, being light touch and respectful of the autonomy of the teams was a factor that underlined any decisions taken: *“we are here as more of a choreographer, or an orchestra leader to help you reveal the strengths of the things you’re building and engender the conversations to happen”*.

### 4.2 Improving Prioritisation

The second change concerned how the organization prioritized work items coming from senior management, and integrated them into the development teams’ work. The example provided by our participants was that the company received an external demand regarding their compliance with a new regulation. This demand needed to be prioritized against other work at the organizational level. The compliance request came in when the teams were *“6 to 9 months away from launching 3 to 4 major games”*.

To respond to this external request, they broke it down into discovery chunks so it wasn’t *“all or nothing”* but became a set of tractable scenarios to assess and prioritize



value: “*make smaller pieces of value or smaller avoidance of pain*”; “*the discussion quickly boiled down to [...] what is the most clear value*”; “*we would cycle through those [...] topics until [...] we had an alignment on an absolute rank of [...] priorities*”. The change was to approach the problem from a discovery perspective.

### 4.3 The Factors Considered

To answer our research question we looked for the main factors that were taken into account when making these changes.

The themes that emerged from the analysis of data around the first change were about prioritization, and negotiation of common work (“*we turn over all the asks and centrally [...] we do a prioritization*”; “*we now broker discussions between the people that asked for the work*”; “*it’s very much a combination of a bottom-up and a top-down negotiation that happens*”). The organization had to find a way that did not disturb the teams, was light touch, and also brought the teams along (“*we saw it as a choreography of behaviours*”).

For the second change, responding to external events by the top of the organization led to work breakdown; this needed to take into account business value, users and risk to the organization. “*born a prioritization system [...] pull all the work and all the asks[...]by engaging in this central ritual, this central overhead which gets us no value to the players and no output from your teams the value you get is stability.*”

Achieving business value and stability through prioritisation was a slow process requiring negotiation of common work and brokering discussions across the organization (“*focusing on choreographing all the work towards a single purpose*”; “*So in this way we, kind of slowly and iteratively, pulled detail out from the depths of teams, and teams of teams and started to reveal those and have discussions. In doing so, we slowly evolved standards, [...], and in that way we didn’t come in, and just say ‘thou shalt fill out this form with all this detail and then we’ll disappear into our ivory tower and tell the customer what’s going on’*”).

Having these conversations eventually led to a state where there was alignment, knowability and transparency across teams that could support predictability for central business (“*finding the balance between empowering their agility and their best practices and giving the [...] large organization value*”; “*the value in an organization is not the number of great ideas it has, it’s ability to win all those and make sure only the most valuable ideas actually come down to the teams [...] so giving visibility into that and enforcing the right conversations far enough ahead in the executive team*”; “*going back to the balance of value versus predictability, what we were trying to do is negotiate for the internal customer, the game team, and align[...] on what were the major chunks of things they needed*”).

The themes identified from analysing the whole data set are shown in Table 2. Some of these factors resonate with the themes identified by practitioners when asked about sustaining agile [18], e.g. the focus on business value and user need, and being independent in the sense of respecting teams’ autonomy. Other factors relate specifically to the agile philosophy, e.g. being lightweight and encouraging discussion. Yet others consider the organization as a whole and the need to maintain a common goal, e.g. predictability and alignment.

The results highlight three perspectives that AgileCo took when making changes to sustain agility: an organization-wide perspective; an awareness of the external environment; and, an understanding of how external changes affect its agile teams. But sustaining agile is complex and this kind of analysis does not provide practical guidance and so we sought to extend our interpretation by investigating theoretical lenses.

**Table 2.** Factors considered when making changes to sustain agility

Factor		Quotes
Users and business value	Business value	“the work system balancing value versus predictability”
	Knowing users/players (importance of players)	“AgileCo is a very player focused company, and anything that leaves some of our players out, no matter how small a slice of the overall pie they might be, is something we take to heart and definitely want to solve”
Risk	Risk	“so the outcome was we aligned we will accept the risk, we understood the risk”
Not disturb the teams	Reduce pain/overhead	“value is two things right, avoiding pain and adding goodness” “we would lighten that overhead”
	Not disrupt coal face while also having predictability	“...game ABC can have some predictability about ...”
	Maintain autonomy	“allowed them to have varying work systems” “allowing that total ownership to continue to flourish”
Predictability	Predictability for central business	“pivot from highly agile to highly knowable state” “so that we could be more predictable” “but also gave us predictability and an ability to have a conversation”
	Transparency	“Transparent in that it is like the work itself radiates obviousness “

*(continued)*

**Table 2.** (continued)

Factor		Quotes
	Knowability	“knowability is about the kind of ease of transitioning into, like, a new team member or a new team joins and someone points and says this is the system we use that they just kind of melt into it without, you know, with ease.”
	Alignment	“Are we aligned with our customers” “we would cycle through those topics until the room felt we had an alignment”
Collaboration	Negotiation of common work	“the purpose of this ritual that we engage in quarterly is quite frankly to have those discussions to reveal all of that opinion and try and piece together a path forward and in that way”
	Conversation/discussion	“we would have this conversation” ... “very good discussions are ensuing” “while it was a very rich conversation”
	Prioritisation	“<was> born a prioritization system [...] pull all the work and all the asks[...]by engaging in this central ritual, this central overhead which gets us no value to the players and no output from your teams the value you get is stability
	Breakdown	“gives us a richer understanding and a better breakdown of the work.”

(continued)

**Table 2.** (continued)

Factor		Quotes
Light touch	Lightweight	“So we started very lightweight”
	Choreograph not direct	“We saw it as a choreography of behaviours, hey if you have concern this is how you express it, so we discuss it, if you have completed your work, this is how you mark it so that it appears in the system, if we need to engage with the customer and we feel there is a gap in their acceptance, or in a set of features that we think solves their ask, this is how we engage in that discussion.” “focusing on choreographing all the work towards a single purpose”; “we are here as more of a choreographer, or an orchestra leader to help you reveal the strengths of the things you’re building and engender the conversations”

## 5 Discussion: Three Theoretical Lenses

We consider three theoretical frameworks that reflect the three perspectives identified in our study, and explore whether they may help extract practical insights from our findings: paradox theory, situation awareness and shared mental models. These three were chosen as they have been used in previous studies of agility, and because they resonate with the changes AgileCo undertook, and the factors they considered.

Paradox theory [7] helps organizations to navigate the complexities of decision-making in the context of everyday contradictions. It has been used to understand the tensions encountered in organizational transformation to agility [2] – an organization-wide perspective. Situation awareness is “the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future” [26] – which helps organizations understand the external environment and how it may affect them. Shared mental models “are team members’ shared, organized understandings and mental representation of knowledge about key elements of the team’s work environment” [27] – this provides an internal view of the impact of change and how external changes affect agile teams. Each of these lenses is explored in more depth below.

## 5.1 Balances and Paradox Theory

Strode et al. [2] argue that the tensions organizations faced in agile transformations are dilemmas (i.e. choices) that needed to be addressed at the beginning of a transformation, and paradoxes [7] that persisted beyond transformation and need to be balanced continually. The authors identified 13 tensions from three case studies, and suggested questions that leaders should answer to address these tensions. Paradoxes need attention and ongoing negotiation as part of the continuous improvement of the organization, and hence will be at the core of efforts to sustain agility.

### Balances and Paradoxes in AgileCo

The two changes reported by AgileCo and the factors they considered characterise ongoing paradoxes in their desire to sustain agility.

The first change, the introduction of a new work system, resonates, in particular, with the paradox '*Distributed authority vs macro-level goals*' as identified by Strode et al. [2]. AgileCo, when changing the work system, was faced with the question of 'how do we guarantee the teams feel empowered and yet understand the organization's goals?'. Their way of addressing this tension was to balance not disturbing the teams and being light touch with achieving predictability for the organization; this was achieved through collaboration and choreography, putting together quarterly meetings and being explicit about not wanting to disturb the teams.

The second change does not resonate so clearly with any of the paradoxes highlighted in Strode et al. [2]. Instead, it is an example of balancing business-as-usual and business value with the need to address an external demand. We suggest that this is a new paradox that was not encountered in the cases studied in Strode et al. [2]. AgileCo was faced with the question: 'how do we keep our core business without ignoring unexpected external regulatory requests?'. They addressed this by breaking down the external request into discovery chunks that could be tractable and prioritized.

Using balances and paradoxes allowed us to confirm and extend by one the paradoxes and associated questions suggested in Strode et al. [2]. Using this lens to explore sustaining agility, gives an organization-wide perspective in the identification and understanding of paradoxes related to sustaining agility, and associated questions.

## 5.2 Situation Awareness (SA)

SA has been used to understand performance under difficult conditions [8], especially operational contexts such as piloting aircrafts, air traffic control, power plants, advanced manufacturing and medicine, where awareness of context is crucial for the tasks to be performed. Endsley [26] defines models for SA in a dynamic decision making context; SA is separate from decision making and consists of "perception of elements in the current situation, comprehension of current situation and projection of future status". [28].

Several extensions to SA have been proposed. While SA focuses on individuals, Team Situation Awareness (TSA) extends SA to include the situation awareness of other team members and the whole team, and shared SA extends awareness across teams [28,

29]. Distributed SA [30] takes a socio-technical view and considers how systems may also have situation awareness.

### **Situation Awareness in AgileCo**

SA is about the cognitive processes followed prior to decision taking. The context of an agile organization may seem very different from the critical contexts within which SA is typically applied; however, there was a high level of SA in understanding how to sustain agile in the face of change, in particular, in sensing the risk and in focusing on users and business value; in particular, knowing users/players is a strong value for AgileCo and its team members, most of whom are also users/players.

In the context of agile, and of AgileCo, the extensions to SA are particularly relevant as they highlight the interdependence and importance of SA between team members, between teams and across the organization. This was particularly the case when dealing with alignment, predictability and transparency in AgileCo; with all teams being aware of the priorities of others, and being aware of the organization's goals and how their priorities fitted with these goals. Both of AgileCo's changes relate to a move from team situation awareness to inter-team and distributed situation awareness. Within an agile team, situation awareness is maintained through various mechanisms including Scrum board, stand-ups and collective ownership [31]. Inter-team knowledge sharing can be supported through similar mechanisms to a degree [32] but AgileCo faced larger scale issues and needed to make collective decisions. In their case, distributed situation awareness needed to be established so that sensible decisions could be made. This relates in particular to the factors predictability and collaboration. However, SA pertains "in the moment" or over a timescale that is finite. For example "DSA can be defined as activated knowledge for a specific task within a system"[30]. In AgileCo, and for other situations of agile sustainability, this would have to be embedded in order to keep the awareness up-to-date and relevant.

Researchers have been looking into what helps develop TSA and shared SA; Bolstad and Endsley [33] mention "requirements, devices, mechanisms and processes". For AgileCo, as an agile organisation, these were present; team members were made aware of information needed by others through predictability and transparency (requirements), "devices" such as communication and collaboration were also present as were effective team processes. One mechanism they mention, shared mental models, is discussed next. TSA and shared SA are important in sustaining agile, in having a good perception of the external environment. Using this lens to explore the factors being considered during change may help organizations determine how well they are supporting situation awareness in their bid to sustain agility.

### **5.3 Shared Mental Models**

Shared mental models theory [9] explains how teams adapt to change and cope with changing demands [34] when they develop shared mental models. Schmidt et al. [27] use team adaptation theory [35] – "(1) assess situations appropriately and build a coherent understanding of a new situation, (2) adjust their plans accordingly, (3) coordinate their work to fit the new situation, and (4) learn by evaluating their effectiveness" – to propose

that “team adaptation is positively related to a high sharedness of team members’ mental models.”.

### Shared Mental Models in AgileCo

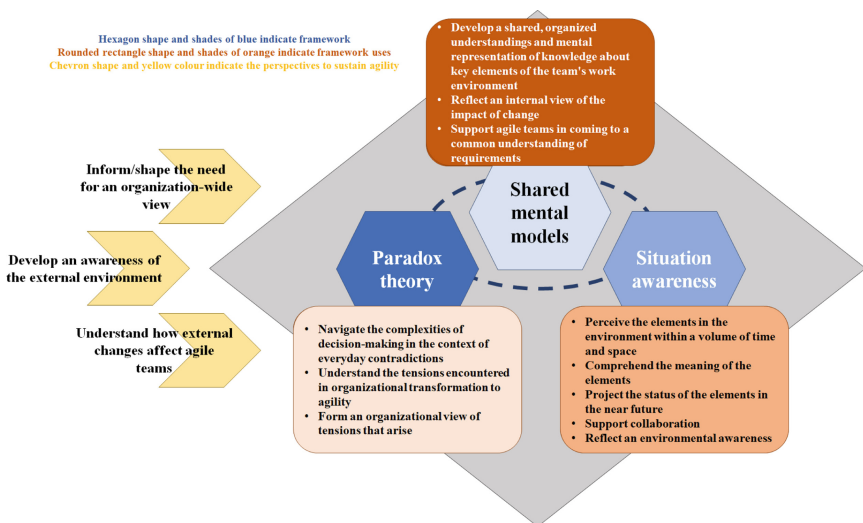
The way teams are empowered, autonomous and effective in AgileCo indicates that there is a high level of understanding and sharing amongst team members both regarding the tasks undertaken and the way the teams are organized.

The factors considered when dealing with the two changes made to sustain agility, indicate the importance of shared mental models at the team level, for example, the imperative to not disturb the teams indicates a shared understanding of how the teams are structured as being effective. The factors considered during these changes highlight that shared mental models also needed to stretch across teams; for example, negotiation of common work, conversation/discussion under collaboration, as well as alignment, knowability, transparency, and predictability for central business to guarantee predictability for the organization. These are important ingredients in achieving a shared understanding [9, 27].

It is also worth noting that, in AgileCo, negotiation and discussions happened both within the work system but outside it too. Shared understanding was supported in the processes and technical context but also needed to extend to the business context. This negotiation outside the work system relates to “articulation work” defined by Strauss as “the specifics of putting together tasks, task sequences, task clusters – even aligning larger units such as lines of work and sub-projects – in the service of work flow” [36].

With shared mental models consideration of sustaining agile needs to go beyond their original focus on individuals and teams, and focus on the wider organization and across teams. Using this lens to explore the factors being considered during change may help organizations ensure that shared mental model practices are supported both within teams and across the organization, hence leading to better shared understanding.

Figure 1 summarises the three theoretical frameworks and how they support different perspectives in sustaining agile.



**Fig. 1.** The three theoretical frameworks support different perspectives in sustaining agility

## 6 Conclusions

### Limitations

A key methodological issue for CDM studies relates to how closely the interviewee was engaged in the events described. In our study, the first interviewee was responsible for deciding the change needed, and in designing the new work system. S/he also was involved closely in prioritization improvement. In this sense s/he personally experienced the decision-making. The other two interviewees provided different perspectives on the situation and were also personally involved in the development of and working within the resulting system.

The data was collected by the two first authors, who also carried out the analysis and discussed it with the research team. The findings of this study have been presented and discussed with key members of the participating team.

This is an initial study focusing on one agile organization and the factors considered when it needed to sustain agility. The data collected and the conclusions drawn directly from it are therefore limited. But our aim was to trial a different data collection approach and analysis frameworks that could be used to research this phenomenon. We used CDM, a novel approach to data collection, and investigated how three theoretical lenses may be used to interpret the resulting data to uncover deeper insights and practical consequences. Our intention was exploratory aiming to identify possible ways for further research in an area that has not been widely researched.

### Lessons Learned

In this work we used CDM as the data collection framework together with thematic analysis to identify factors that organizations consider when making changes to sustain agility. We then explored three theoretical lenses to see what kind of insights they may yield when interpreting the findings.

Using CDM supported our goal of finding the detail and specific examples of changes that AgileCo had to make. The multi-pass retrospective approach elicited detail, but was challenging to apply because of the perceived repetition from our participants. Although the questions targeted different aspects of the change at different times in the enquiry the same issues were visited repeatedly. Interviewing people from different roles provided different perspectives on the same events. We would recommend others to use this approach, but encourage them to train interviewers and prepare interviewees for the cyclical nature of the interrogation.

A thematic analysis identified factors that were being considered during changes needed to sustain agility, and based on this, we identified three potential theoretical lenses to extend our findings further. For each of these we have demonstrated the kind of practical insights that could be extracted from the data. These frameworks illuminated three key dimensions relevant to sustaining agility: teams **vs** organization, understanding the environment **vs** understanding **how** to deal with its impact inside the organization; and understanding the now **vs** looking into the future.

Based on this initial study we believe that approaching an investigation of sustaining agile in this way can yield interesting and practical results.



**Acknowledgments.** We would like to thank our participants. We would like to acknowledge the funding from Agile Business Consortium and the support by the Business Agility Institute. We would also like to thank Diane Strode with whom we had fruitful discussions about the theoretical lenses chosen.

## References

1. Freudenberg, S., Sharp, H.: The top 10 burning research questions from practitioners. *IEEE Softw.* **27**(5), 8–9 (2010)
2. Strode, D., Sharp, H., Barroca, L., Gregory, P., Taylor, K.: Tensions in organizations transforming to agility. *IEEE Trans. Eng. Manag.* **69**(6), 3572–3583 (2022)
3. Gregory, P., Barroca, L., Sharp, H., Deshpande, A., Taylor, K.: The challenges that challenge: engaging with agile practitioners' concerns. *Inf. Softw. Technol.* **75** (2016)
4. Gregory, P., Strode, D.E., Sharp, H., Barroca, L.: An onboarding model for integrating newcomers into agile project teams. *Inf. Softw. Technol.* **143**, 106792 (2021)
5. Shull, F.: Who needs evidence, anyway? *IEEE Softw.* **24**(5), 10–11 (2007)
6. Hoffman, R.R., Crandall, B., Shadbolt, N.: Use of the Critical Decision Method to elicit expert knowledge: a case study in the methodology of cognitive task analysis. *Hum. Factors* **40**, 254–276 (1998)
7. Smith, W.K., Lewis, M.W.: Toward a theory of paradox: a dynamic equilibrium model of organizing. *Acad. Manag. Rev.* **36**(2), 381–403 (2011)
8. Klein, G.: Analysis of situation awareness from critical incident reports. In: Endsley, M.R., Garland, D. (eds.) *Situation Awareness Analysis and Measurement*, pp. 51–72. Lawrence Erlbaum Associates, Mahwah (2000)
9. Yu, X., Petter, T.: Understanding agile software development practices using shared mental models theory. *Inf. Softw. Technol.* **56**(8), 911–921 (2014)
10. United Nations. Sustainable Development Goals. <https://sdgs.un.org>. Accessed 3 Apr 2023
11. OED. Oxford English Dictionary. <https://www.oed.com>. Accessed 3 Apr 2023
12. Buchanan, T., et al.: No going back: a review of the literature on sustaining organizational change. *Int. J. Manag. Rev.* **7**(3), 189–205 (2005)
13. Holbeche, L.: Organisational effectiveness and agility. *J. Organ. Eff.* **5**(4), 302–313 (2018)
14. Holbeche, L.: *The Agile Organization: How to Build an Engaged, Innovative and Resilient Business*, 2nd edn. Kogan Page (2018)
15. Miceli, A., Hagen, B., Riccardi, M., Sotti, F., Settembre-Blundo, D.: Thriving, not just surviving in changing times: how sustainability, agility and digitalization intertwine with organizational resilience. *Sustainability* **13**(4) (2021)
16. Senapathi, M., Srinivasan, A.: Sustained agile usage: a systematic literature review. In: *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering - EASE 2013*, p. 119. ACM Press, New York (2013). <https://doi.org/10.1145/2460999.2461016>
17. Sedano, T., Ralph, P., Péraire, C.: Sustainable software development through overlapping pair rotation. In: *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 19:1–19:10 (2016)
18. Barroca, L., Gregory, P., Kuusinen, K., Sharp, H., AlQaisi, R.: Sustaining agile beyond adoption. In: *Proceedings - 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2018, Prague* (2018)
19. Rogers, E.: *Diffusion of Innovations*, 5th edn. Simon and Schuster (2010)

20. Crandall, B., Klein, G., Hoffman, R.: *Working Minds: A Practitioner's Guide to Cognitive Task Analysis*. The MIT Press, Cambridge (2006)
21. Flanagan, J.C.: The critical incident technique. *Psychol. Bull.* **51**(4) (1954)
22. Wong, B.L.W., Blandford, A.: Situation awareness and its implications for human-systems interaction. In: Apperley, M. (ed.) *Proceedings of the Australian Conference on Computer-Human Interaction (OzCHI 2001)*, Perth, pp. 181–186 (2001)
23. Lopez, T., Petre, M., Nuseibeh, B.: Getting at ephemeral flaws. In: *5th International Workshop on Co-operative and Human Aspects of Software Engineering (CHASE)*, pp. 90–92 (2012). <https://doi.org/10.1109/CHASE.2012.6223030>
24. Wong, B.L.W.: Critical decision method data analysis. In: Diaper, D., Neville, S. (eds.) *The Handbook of Task Analysis for Human-Computer Interaction*. Lawrence Erlbaum Associates (2003)
25. Braun, V., Clarke, V.: Using thematic analysis in psychology. *Qual. Res. Psychol.* **3**(2), 77–101 (2006)
26. Endsley, M.R.: Theoretical underpinnings of situation awareness: a critical review. In: Endsley, M.R., Garland, D.J. (eds.) *Situation Awareness Analysis and Measurement*, pp. 23–48. Lawrence Erlbaum Associates, Mahwah (2000)
27. Schmidt, C., Kude, T., Heinzl, A., Mithas, S.: How agile practices influence the performance of software development teams: the role of shared mental models and backup. In: *Thirty Fifth International Conference on Information Systems*, Auckland (2014)
28. She, M., Li, Z.: Team situation awareness: a review of definitions and conceptual models. In: Harris, D. (ed.) *EPCE 2017. LNCS (LNAI)*, vol. 10275, pp. 406–415. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-58472-0\\_31](https://doi.org/10.1007/978-3-319-58472-0_31)
29. Salmon, P.M., et al.: What really is going on? Review of situation awareness models for individuals and teams. *Theor. Issues Ergon. Sci.* **9**(4), 297–323 (2008)
30. Stanton, N.A., Stewart, R., Harris, D., Houghton, R.J., Baber, C., McMaster, R.: Distributed situation awareness in dynamic systems: theoretical development and application of an ergonomics methodology. *Ergonomics* **49**, 1288–1311 (2006)
31. Sharp, H., Robinson, H.: A distributed cognition account of mature XP teams. In: Abrahamsson, P., Marchesi, M., Succi, G. (eds.) *XP 2006. LNCS*, vol. 4044, pp. 1–10. Springer, Heidelberg (2006). [https://doi.org/10.1007/11774129\\_1](https://doi.org/10.1007/11774129_1)
32. Santos, V., Goldman, A., de Souza, C.R.B.: Fostering effective inter-team knowledge sharing in agile software development. *Empir. Softw. Eng.* **20**(4), 1006–1051 (2014). <https://doi.org/10.1007/s10664-014-9307-y>
33. Bolstad, C.A., Endsley, M.R.: The effect of task load and shared displays on team situation awareness. *Proc. Hum. Factors Ergon. Soc. Annual Meet.* **44**(1) (2000)
34. Mathieu, J.E., Heffner, T.S., Goodwin, G.F., Salas, E., Cannon-Bowers, J.A.: The influence of shared mental models on team process and performance. *J. Appl. Psychol.* **85**(2), 273–283 (2000)

35. Burke, C.S., Stagl, K.C., Salas, E., Pierce, L., Kendall, D.: Understanding team adaptation: a conceptual analysis and model. *J. Appl. Psychol.* **91**(6), 1189–1207 (2006)
36. Strauss, A.: The articulation of project work: an organizational process. *Sociol. Q.* **29**(2), 163–178 (1988)



**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# The Role of Responsiveness to Change in Large Onboarding Campaigns

Darja Smite<sup>1,2</sup>  and Nils Brede Moe<sup>1,2</sup> 

<sup>1</sup> Blekinge Institute of Technology, Karlskrona, Sweden  
darja.smite@bth.se, nils.b.moe@sintef.no

<sup>2</sup> SINTEF Digital, Trondheim, Norway

**Abstract.** Onboarding is a process of organizational socialization of the new hires, that includes recruitment, orientation, training, coaching and support. While onboarding individuals into an organization is a rather straightforward task, little is known about 1) onboarding hundreds of developers and 2) doing it on a distance in outsourcing situations. Furthermore, the subject of sustainable growth with respect to organizational capabilities and culture is often overlooked. This paper reports findings from an exploratory multi-case study of two large onboarding campaigns. We collected empirical data from interviews, retrospectives, onboarding documentation and onsite visits. Based on the empirical study, onboarding hundreds of software engineers in a complex agile product development environment which lacks documentation and puts high demands on engineers' knowledge and skills is a challenging and costly endeavor. To save the costs and for practical reasons, large-scale onboarding is organized in batches with the first batch trained onsite, and the later batches trained internally. We report challenges faced in the two cases and discuss possible solutions. One core finding is that a good plan combined with the organizational agility, i.e., the responsiveness to change, together with organizational maturity determined the success of organizational scaling. The presented cases contribute to the scarce research on knowledge transfer and onboarding in a large-scale agile context.

**Keywords:** Onboarding · Scaling · Training · Teams · Large-Scale Agile · Software Engineering · Empirical · Case study · Sustainable Organizational Growth

## 1 Introduction

To satisfy the need for new features and services at a faster speed, agile software companies face the need to scale their development capacity. However, organizational scaling carries a number of challenges. The first challenge occurs when a company needs to scale fast. Since employing many competent engineers with technical and domain expertise might be difficult and costly, many opt for university graduates and junior engineers instead considerably lowering the competence level and diluting the organizational culture, which also negatively impacts overall performance and quality. The second challenge relates to the difficulty of employing local engineers [1]. As a result,

many companies “scale out” by either teaming up with external suppliers nationally and internationally or globalizing via acquisitions and setting up own subsidiaries. However, barriers such as domain complexity, differences in processes and even in the use of agile processes (e.g., timeboxed vs. flow-based development), and employee turnover all create challenges when scaling out [2]. Further, cultural barriers have been found to be impediments to agile software development practices [3]. One particular significant factor leading to success when scaling is effective onboarding [4, 5].

Onboarding is the process of supporting new employees regarding their social and performance adjustment to their new job [4]. It is also known as organizational socialization. In software development companies, onboarding of new developers happens to replace retired developers, to replace developers who left or will leave, to replace existing developers who changes roles, to offload existing developers or increase capacity, or to incorporate new people with unique expertise who bring new ideas and thus help a company to innovate [5].

Traditionally recruitment has been studied as a process that happens within corporate boundaries. The meaning of the “onboarding” process is thus commonly related to facilitating the transition of the newcomers from being organizational outsiders to being insiders [6]. Onboarding can also happen in a project or a team by relocating already employed staff. Large emphasis in traditional onboarding is put on the social integration into a team and ensuring mentoring and support from teammates. When onboarding an individual to replace a team member who retires, takes up a new role within the company or leaves the company, training is often performed by the person who is to be replaced, or by immediate teammates. Onboarding in general is a well-researched field with studies of diverse type of professions [4, 7–10], and some empirical studies about onboarding newcomers in software companies [5, 11–13] as well as recent interest in the context of agile software development [14].

Large-scale agile involves hundreds of engineers and tens of teams spread not only geographically but also across legal boundaries of a firm. Onboarding in such contexts may thus require recruiting and socializing engineers at a remote subsidiary or at a sub-contractor, which is obviously a challenging endeavor. The reasons for this are twofold. First, newcomers need to be integrated not only into the boundaries of the subsidiary or the outsourcing supplier, but also into the assignment or the project led by the organization. Second, all recruitment and onboarding activities typically happen on a distance, which is not a straightforward task with little available knowledge on the topic [5].

In this paper, we present two empirical cases of large onboarding campaigns in the context of global agile software engineering and discuss the practices that can facilitate the onboarding. Our research is driven by the following question: **What are the strategies for large onboarding campaigns?**

## 2 Background

Onboarding models suggest that organizational socialization usually starts with recruitment and continues with orientation, training, coaching and support [4]. While the recruitment and orientation can be viewed as an entrance to the company, training focuses

on the readiness of the newcomers for the particular job within the boundaries of, e.g., an assignment, or a team. Similarly, training is given to existing company employees, when they are transferred to new assignments within the company. Coaching and support are common means to help employees climb up their learning curve and improve performance. Just like agile coaches, engineering coaches paired with new hires target performance strategies by coaching to improve work processes, and by teaching new techniques and providing technical guidance [15]. Training as well as coaching and support during the onboarding process are the two well-researched areas that in the area of software engineering are primarily researched in open-source projects [13, 16–18]. Further, research shows that providing support to new hires plays even a more prominent role in onboarding success, than training [13].

Success of onboarding is often associated with the extent to which an organization succeeds to ensure that new employees possess different knowledge and skills [4]:

- Basic legal and policy-related rules and regulations,
- Understanding of their new jobs,
- Grasping organizational norms and culture,
- Establishing the necessary interpersonal relationships and information networks.

Social integration has recently gained attention. Onboarding is said to be more successful when the assimilation and integration of the new hires at the new job is assisted through educating them about the cultural norms, clarifying the expectations, helping to recognize formal and informal channels and assisting the integration into established communities [4, 9, 19]. Strong social connections are also said to improve productivity long-term [20]. In addition, in an agile environment the new hires might need to also adjust their behavior to the agile values and undergo a mindset change [14].

The emphasis, formality and duration of the different onboarding functions, may differ in different recruitment situations. For example, the job specificity and the entry competence of a newcomer may require less or more training, mentoring and support. Similarly, previous experience and expertise, whether specific for the job, related and or even unrelated may determine the amount of orientation and training [21]. Further, the amount of the simultaneous new hires plays an important role in determining how onboarding is organized. Large recruitment companies in agile companies tend to have more formalized onboarding functions, ensure extensive training and dedicated mentoring and support [5]. However, little is known about the practical aspects of large onboarding campaigns, especially in agile and global software engineering contexts.

### 3 Methodology

In this paper, we report our findings from an exploratory multi-case study of GlobCo and its collaboration partners. GlobCo is a large international company headquartered in Northern Europe that develops a wide range of software-intensive products and solutions, including generic software products offered to an open market and complex compound systems with customized versions. Agile ways of working including Scrum have been practiced since 2008, and as part of adopting agile methods, communities of practice (CoPs) were introduced early. The company has been globally distributed for a long

time with many own sites across the world, and a network of external vendors. Here, we study two cases of large onboarding campaigns in outsourcing assignments for their two different vendors, namely Case A with ConsultCo from Poland, and Case B with SupplyCo from Croatia. All company names are pseudonyms to preserve confidentiality.

Our case study is a part of a larger study, the main focus of which is on scaling organizational efforts in agile companies. In the large study, we have researched individual onboarding cases of different size and in different contexts, including vendor switching [22] and knowledge transfer. The cases selected for the present study are all cases of onboarding hundreds of software engineers.

Our study follows the guidelines of the case study design proposed by Yin [23]. Our unit of analysis, the “case”, is the onboarding of new software engineers at the outsourcing supplier side in the context of a client-supplier assignment. The goal of our exploratory study is to find out what is happening, seeking new insights, and generating recommendations based on the state-of-the-practice [24].

**Table 1.** Overview of the cases and data collection activities.

Case	Company	Interviews	Onsite visits	Documentation
Case A Onboarding of 150 engineers in 3 batches	GlobCo, Sweden and Canada	5 individual interviews	No	Assignment specification, follow up assessments and progress reports
	ConsultCo, Poland 1	2 individual interviews, retrospective (9 engineers and managers; 1,5h), group interview (7 engineers; 1h)	Yes	
	ConsultCo, Poland 2	2 individual interviews, retrospective (6 engineers; 1,5h), group interview (4 engineers; 1,5h)	Yes	
Case B Onboarding of 200 engineers in 3 batches	GlobCo, Poland	6 individual interviews	No	Assignment specification
	SupplyCo, Croatia	3 individual interviews, retrospective (9 engineers; 1,5h), group interview (7 engineers and managers; 3h)	Yes	

### 3.1 Data Collection

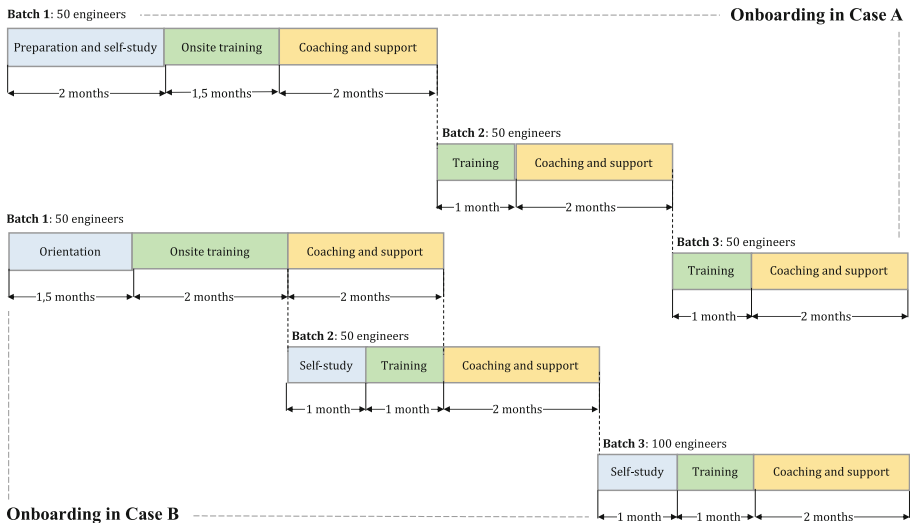
Our exploratory research is qualitative in nature. The data collected to answer our research question was collected through qualitative interviews involving individuals and groups and onsite observations (see Table 1 for details). We performed twenty-two semi-structured interviews with twenty-seven interviewees in total across all cases. The interviewees were selected by the companies and were set to represent the key roles in the onboarding campaigns. In both cases, we have interviewed the assignment owner from GlobCo, and two to seven representatives from the vendor companies. During the interviews, we asked the interviewees to recall the events of the transfer, and reflect on what went well, what did not go that well, and what recommendations could they give for future transition projects. All individual interviews were 45–60 min long, conducted via Skype for Business teleconferencing tool, while the group interviews and retrospectives varied in duration (see Table 1), organized at the outsourcing suppliers onsite in Poland and Croatia. Individual interviews were recorded with the consent of the interviewees using AudioNote (special software tool for recording and note-taking) while group sessions were led by one researcher and closed to live-transcribed by the second researcher. Some group sessions were organized as a retrospective session (see Table 1). To elicit the relevant problems, we chose to use an exercise where participants discuss issues that need to be dropped, added, kept or improved (DAKI). This exercise is suitable when there are many participants and issues. We then facilitated a discussion of the most urgent items by using the Lean Coffee technique.

In addition to the qualitative data, we received transition project documentation, containing detailed assignment specifications and knowledge transfer documentation. Assignment specifications were used to understand the assignment details, parties involved, and the amount of the work being transferred. Detailed transfer plans and personnel profiles were used to assess the scope of each onboarding project and associated activities. Travel plans were used to check which onboarding activities were carried out in co-location and how many people were given this opportunity. Progress reports were used to elicit the challenges and obstacles, as well as the carried mitigation actions reported during the onboarding campaigns.

### 3.2 Data Analysis

The data analysis was organized in iterations. The preliminary data analysis was performed upon the data collection. The interview protocol was used to capture the responses through simultaneous notetaking [25]. These initial “jottings” and summaries were later expanded based on the audio recordings whenever necessary. During the next stage, we used the preliminary coding to draft the case histories, which focused on the factors that hindered and factors that facilitated onboarding. We used an initial set of categories based on the risk factors identified in earlier research on knowledge transfer [26], adding new factor that emerged in the analysis. The case histories were then sent for validation to GlobCo’s outsourcing partners, ConsultCo and SupplyCo, and later presented and discussed with the representatives from the client organization, GlobCo. Finally, we created the narratives (presented in Sects. 4.1 and 4.2) by revisiting the interview transcripts





**Fig. 1.** Onboarding activities organized in three batches in Case A and Case B.

and notes from the group interviews and retrospectives, as well as the onboarding documentation. To explain what was going on during the onboarding, we also reconstructed the onboarding activities (Fig. 1). To provide the traces between our interpretation of the events and the data, we illustrate the core findings with transcribed quotations from the interviewees or the in-vivo notes taken during the group interviews and retrospectives.

### 3.3 Limitations and Threats to Validity

As any case study research, our study has several limitations [23]. Our findings are subject to interviewee bias, which we addressed by involving interviewees from all participating companies (data triangulation) and using progress reports (method triangulation). To increase the reliability and avoid false interpretations, we sent the draft of our report for review by key informants. Our study is also strongly bounded by the context of the studied companies (company sizes, size of the onboarding campaigns, outsourcing relationships involved) and the employed onboarding approach (three batches with relatively aggressive schedules), limiting the generalizability of the findings [23]. To support the readers in judging the applicability of our findings in their contexts, we detailed the case descriptions as much as possible. We believe that the findings can be of particular interest for managers involved in large-scale software development of complex legacy products facing the need for onboarding large numbers of engineers.

## 4 Insights into Large Onboarding Campaigns

### 4.1 Case A: Onboarding 150 Engineers

In ConsultCo, we studied a large recruitment and onboarding campaign for the new assignment by GlobCo that required 300 engineers to be onboarded as a replacement for the GlobCo internal engineers in Sweden, Canada and China. ConsultCo bid contained an offer of 170 engineers in China and 150 engineers in Sweden (the latter is the focus of our study). However, due to recruitment difficulties, ConsultCo could only hire 20 engineers in three different sites in Sweden while the vast majority (130 engineers) were hired in two subsidiaries in Poland.

Due to the scale of the assignment (150 engineers), the onboarding was organized in batches (see Fig. 1 – Case A), because organizing onsite on-the-job training for so many engineers simultaneously was regarded as practically impossible. The first onboarding cycle started with a 2-months long preparation and a period of self-study, followed by the onsite training phase, and coaching and support. The supplier company tried recruit experienced engineers from a GlobCo's competitor, who were later said to have much faster onboarding. Unfortunately, only half of the people could travel for the onsite training, so there were groups of engineers learning from the video recordings.

The training and knowledge transfer was kicked-off with a four-day long program in Sweden, and further activities were organized on-site in either Sweden or Canada. Admittedly, the GlobCo engineers were working under much stress and market pressure and could not dedicate due attention to the trainees. In retrospect, the time dedicated for onsite training was also regarded as insufficient. The reasons were threefold. First, the onboarding schedule generally was very aggressive (5,5 months for the first batch and 3 months for each of the following batches) due to large market pressure. Second, the product concerned was recognized as “extremely complex” with very large product areas and hardware and software architecture differing from competing products in the same area. Therefore, the threshold for learning the product was very high and the learning curve for understanding the system end-to-end was estimated to take 1–2 years. Even becoming at least somewhat productive according to the newly onboarded ConsultCo engineers was said to take 6–12 months. Finally, the product was insufficiently documented, especially concerning the legacy parts and new functionality. Besides, some material was outdated, resulting in the false feeling of obtaining knowledge. As described by an engineer:

“[A colleague], who newly graduated from the university, was talking about this shock. He was reading the material and looking through presentations in the beginning, and he said [the company] has a lot of material and documentation, but there were two main challenges. First, the language [the company] has [includes] a lot of abbreviations. [...] And he had a hard time understanding the material. And then the second challenge was the outdated material. [...] He read it and he thought he has gained a lot of information. And then later when he saw the code he saw that the code differed.”

Another engineer explained the challenge of understanding the product:

“My feeling is that still the best source is the code itself. [...] But there are still millions of lines of code. So it’s not possible to read the whole code”.

And another engineer estimates the time it takes for individual exploration:

“It takes 1 hour for an experienced guy, but you need 1-2 weeks to figure it out on your own”.

Nonetheless, GlobCo expected that by the end of the first onboarding cycle, engineers from the first batch will be able to further introduce and train the new teams locally with a minimum support from the original engineers in GlobCo.

In the following batches, several changes were made in the onboarding program. First, the self-study phase was skipped. Although the live trainings have been recorded, watching the trainings was not a popular learning activity. As an engineer explained: “People did not like reading material and watching videos for 2 months”. Instead, the self-study parts of the training occurred in parallel with the integration in the teams and were based on the customized material adjusted to the local needs created by the engineers from the first batch. As an engineer explained:

“In the next few months we prepared a checklist including mandatory training steps every new project member has to go through in order to get the knowledge because a lot of people that are joining our project don’t have any [domain] background... There were some materials created during our internal trainings especially recorded trainings in Polish. It is always a little bit easier to work in your native language when talking about complex things”.

Integration in the teams was organized by splitting the teams onboarded in the first batch (e.g. 10 teams of 5) and creating twice as many teams in the second batch (e.g., 20 teams of 5), and further increasing the number of teams in the third batch (e.g., 30 teams of 5). However, this meant that half of the engineers from the first batch had to change roles or product areas they worked with, resulting in the training efforts partially wasted. As an engineer explains:

“In fact, some of the people who went to Canada or Sweden spent time on learning one area and then they were moved to another area. The area they received the knowledge about [during onsite training] didn’t really match the features that they have received. [...] People had to be spread somehow. [...] I don’t know if we really could have avoided that”.

This was one of the reasons why competence build-up was slower than expected and the level of experience in some areas was notably low, as noted in the progress reports. Besides, the very lack of direct knowledge transfer from the original developers in this case had a prominent impact, as an engineer describes:

“We can compare what type of onboarding [a colleague’s name who was trained onsite] received and others who have watched the presentations. More should be sent for onsite training, but not necessarily all.”

Onboarding in Case A had a mixed success. Despite good cooperation, initial enthusiasm and motivation to engage from the SupplyCo side, GlobCo did not have the time to organize efficient onsite training and upon the completion of the onboarding the speed of execution, delivery quality and ability to take full responsibility for assigned tasks was below expectations. There were other factors that contributed to this too. For example, ConsultCo organization in Poland was relatively new and thus did not have a safety net of colleagues that could support the newly recruited batches. Besides, the management acknowledged that the very complexity of the assignment from the start was underestimated and employed too many inexperienced people without the domain and process-related knowledge, or prior experience with working for GlobCo. Due to the large scale of the recruitment campaign, ConsultCo focused primarily on ramping up the number of engineers compromising on the competence front. Finally, the long periods of low productivity and other typical career changes led to a substantial amount of resignations causing significant damage on the capability front.

## 4.2 Case B: Onboarding 200 Engineers

In SupplyCo, we studied a large recruitment and onboarding campaign for the new assignment by GlobCo. Prior to the studied assignment, SupplyCo had extensive experiences in collaboration with GlobCo and thus had existing experience with their ways of working. The assignment overall was related to continuous development and maintenance of a very large and complex system, which had already involved over 1000 engineers from six different organizations and was said to take time to learn. The new assignment required onboarding of 200 engineers, which was similarly to Case A organized in three batches (see Fig. 1 - Case B).

The schedule chosen for the onboarding was rather aggressive due to market pressure (4–5 months for each batch and one year for the entire scaling). To ensure successful onboarding, SupplyCo's decided to organize onsite training at GlobCo in Poland for the first batch and put forward a group of highly-experienced inhouse engineers with related expertise and the right attitude – “not shy to push and motivated to get into work”, who can put 110% effort to secure performance. Knowledge transfer started with the general presentations prepared by the client organization, but quickly shifted to real tasks and close integration into the teams of original engineers at the client site, whenever possible. A product owner who did not join a Polish team compares experiences with his colleagues:

“A decision was made that the second team [of trainees] will split and join the existing Polish teams and they will work together with the Polish guys and try to learn as much as possible [...] And at the end, it turned out, I think, that those guys learned much more than the guys in my team that consisted of only the Croatian guys. That was the learning that guys that were part of the mixed teams learned much more than the guys that were left to learn on their own through support”.

This is echoed by an experienced engineer:

“Some developers were integrated in different Polish teams, but my team was assigned a new feature. We could ask for help, but we did not have a person

in Poland and zero experience. We had someone to talk to, but that person was not working on the same feature. There is a steeper learning curve when you are integrated in teams”.

To maximize the learning, some engineers rotated from team to team to be exposed to different parts of the product and different points in the lifecycle of a task. At the end of each day, the SupplyCo engineers organized daily stand up meetings to share their experiences (what have one learned) and follow up on any emerging issues (what challenges one faced). The first batch was deliberately assembled by fast learners who were expected to pass their knowledge to further batches. Besides, their competence has helped to overcome the challenges faced during the knowledge transfer, as engineers at the client organization had themselves limited experiences, as they have been onboarded into the assignment 1–2 years prior to the scaling endeavor. The challenge in Case B was similarly to Case A the product itself. As an engineer describes:

“There’s quite a lot to learn. That’s the biggest challenge, to learn the product. And it is still a big problem. After 7-8 months, we are still learning the product”.

Due to onsite training being “a big investment”, as recognized by the GlobCo stakeholders, the second and third batches have been onboarded by the representatives from the first batch at the supplier site. However, onsite training was also available for engineers in the other two batches, on request. At the end of the first onboarding cycle, the management ran a retrospective to improve the onboarding for the following two batches. For example, there was a need to have a more personal onboarding, and so the joint orientation phase was replaced with the tailored self-study period adjusted for different roles, product areas and personal experiences. As a manager said:

“Not all are going through the same path and pace”.

Further, engineers from the first batch created learning guides for different learning groups (with a detailed agenda), video-recordings and detailed mind maps with the necessary competences and skills reused for the following batches. An engineer explains:

“Later we actually created some presentations and made maps and also created learning tasks so that you can show people how to go through a task and then discuss after they tried to learn all what is needed in order to work. And one month later [the following batch] actually start to work with the team”.

SupplyCo’s training philosophy is described by a manager as follows:

“With documentation it’s not possible to address everything, so you need to figure out what are the crucial things. Then learning methods are also very important. Learning methods [...] must build confidence. [...] So, depending on the setup [...] it’s going to be on-the-job training, different tasks within the team or individually”.

Onboarding in Case B has been regarded as successful. All interviewed GlobCo representatives mentioned good preparation, very good communication between the parties, and experienced people nominated by the supplier as the key success factors. SupplyCo

further describes that they were able to find employees with prior experience of working on GlobCo assignments, and thus ensure process and domain knowledge. The company acted as a safety net, which was not the case for ConsultCo, a relatively new company with many inexperienced new hires. Besides, SupplyCo have shown exceptional motivation to receive the assignment and demonstrate their capabilities.

### 4.3 Cross-Case Analysis: Challenges of Onboarding Hundreds of Engineers

In this sub-section, we report the top five challenges particular for the large onboarding campaigns and the helpful strategies emerging from the two cases.

**Challenge 1: A Large Onboarding Campaign is a Large Investment.** Recruitment and onboarding of hundreds of software engineers in both cases required enormous practical and financial support. Despite the awareness that the best knowledge transfer is organized through on-the-job onsite training, GlobCo could not arrange hundreds of desks and workstations to accommodate all engineers in their premises. Even recruitment of so many engineers can be challenging, as experienced by ConsultCo in Case A. Besides, since the suppliers in our cases were situated on a distance, flying everybody for onsite training required substantial budget resources.

**Helpful Strategies:** To avoid high costs and impracticalities of onsite training for hundreds of developers, companies can organize onboarding in batches. In this approach, the first batch is formed by highly competent and fast learning engineers and sent for onsite knowledge transfer. Upon completion of their onboarding, the first batch shall be able to support onboarding of the following batches.

**Challenge 2: Multiple-Batch Onboarding by Splitting Dilutes the Accumulated Competence.** The batch-based scaling in our cases was organized by splitting. In Fig. 2, we illustrate the onboarding of 54 engineers in three batches, where the first batch comprises three teams trained by the original engineers, which are then split into halves and then scaled with the second batch engineers. The splitting process repeats for the third batch. Every new batch evidently lowers the overall level of experience in the team. If repeating the process each team will have just one engineer trained onsite by the original engineers in the fourth cycle, and after the fifth cycle some teams will not have any engineers trained onsite. In Case B, the third batch consisted of as many developers as in the previous two batches, which resulted in having teams without any first batch engineers. A young engineer from the third batch in Case B explained:

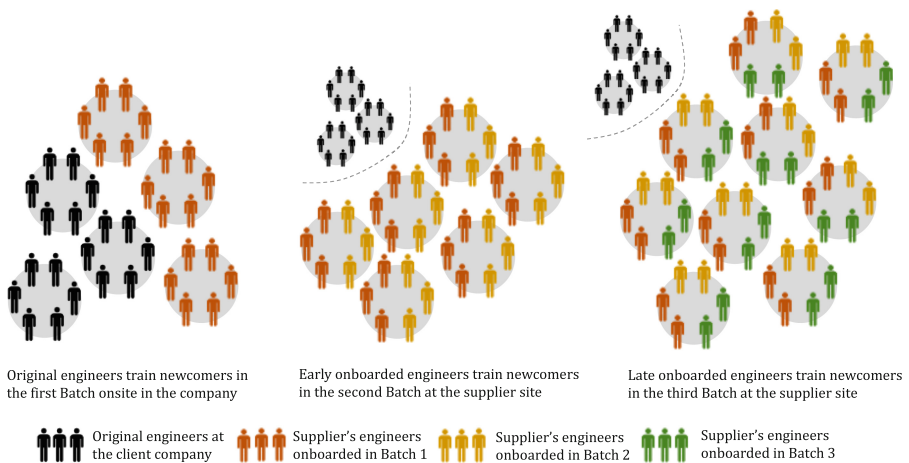
“None of our team members had [domain] experience, none of the team members worked here. We were a team with only new persons. It is not a good practice, although we had a good solution to this – everybody had contacts [with experts] and we had a buddy team somehow. But better to have a mix [of competent and new people].

**Helpful Strategies:** Splitting teams to mix the competent and new people is a useful practice, but with some limitations. When splitting results in not having enough competent people on the team, companies may try pairing the new teams with a “Buddy team”

or a “Sister team”, who then provide the necessary mentoring and support. Buddy teams were practiced in Case A during the onsite training, as described by a manager:

“It’s when [the client company] teams work very close with new teams in [SupplyCo], instead of having 1:1 developer to developer relationship.”

Our results also clearly show that there is no one recipe for team formation and that the decisions shall consider 1) the new hires’ competence levels, 2) the product area selected for onboarding, and 3) the competence levels of the existing teams. Case B representatives mentioned that their strategies varied from extending some teams (which required having smaller teams in the beginning), splitting some teams (especially in the areas that require many engineers), and forming some new teams (especially in the areas that were not covered by the training for the first batch).



**Fig. 2.** An example of the onboarding of ~ 50 engineers organized in three batches by splitting.

**Challenge 3: Multiple-Batch Onboarding by Splitting can Result in Ill-suited Onboarding Efforts.** The success of scaling by splitting depends on the tasks that the onboarded teams receive. The onboarding program of the first batch could not possibly cover the whole system, and thus onboarding was done on selected areas. Naturally, when splitting the initially formed teams, some engineers had to work on something they have not been trained for. As a result, these engineers perceived their training at least partially as a waste. However, as a manager from Case B explained, even when having a good plan, the problem is not always possible to avoid due to changes:

“We had to onboard 200+ people. Can we take half of the product first? That was our plan. But reality was completely different. Once you start [the training], [the scope] starts changing, because the [operational] need is there (points in one

direction), then the need is there (points in another direction), then the need is there (points in the third direction). So, you need to adjust.”

**Helpful Strategies:** One way to address this challenge is to follow Case B example and offer on demand onsite training, for all in the first batch and some in the subsequent batches. Another strategy was to ensure that the self-study parts of the training occurred in parallel with the integration in the teams to make the training relevant (Case A). Alternatively, companies may also consider not forming the teams too early and instead assigning engineers from the first batch to cover all product areas, taking into consideration the additional resources planned in the subsequent batches (in the example on Fig. 2, this would require forming nine pairs of two in the first batch or having more teams in the first batch and fewer later).

**Challenge 4: Poor Onboarding Experiences may Result in New Hires Leaving.** When onboarding employees into a complex software product with poor or no documentation, and especially when employing inexperienced engineers, it may result in long periods of low productivity and confusion, in some cases leading to resignations. In Case A, the resignations among already onboarded employees caused a loss of accumulated experience and expertise. Attrition especially among those trained onsite has devastating impacts since every individual plays an important role in their team, when scaling by splitting the teams. As an engineer from Case A describes his experience:

“A funny situation: someone has made the configuration [of the testing environment]. But he has left, and now nobody knows how to configure it.”

**Helpful Strategies:** Countermeasures for the high levels of complexity and gaps in documentation include recruiting competent engineers with domain and potentially even process experience. This allows to form teams, in which inexperienced engineers are supported by more experienced engineers, ideally with prior engagement with the company. As an engineer from Case B explains:

“We divide the team mixing more experienced and less experienced people. That’s how a lot of companies do this onboarding.”

Other helpful strategies identified in our two cases include on-the-job training, integration in or pairing with experienced teams for mentoring and support, on-demand onsite training, and fostering shared learning within a team being onboarded through regular daily synch sessions and retrospectives to improve onboarding experiences.

**Challenge 5: Disproportional Organizational Growth During Large Onboarding Campaigns can be Devastating.** Turnover in Case A was caused not only by the poor onboarding experience, but also for more typical reasons, including promotions, relocation to other projects, personal reasons. Onboarding in Case B, in its turn, was threatened by the turnover among the trainers on the client side, as a manager describes:

“Attrition can be on both sides. If on the client’s side, you are left without the trainers. In [Case B] we had over 20% attrition. That was a problem.”

In both cases, the success of the onboarding depended on the ability of the companies to respond to changes and compensate the loss or lack of competences primarily through



relying on internal resources. This was possible in Case B, because SupplyCo was a rather mature company, while this was a challenge in Case A, since ConsultCo was a relatively new company and had few resources to help in this situation.

**Helpful Strategies:** The problem of rapid organizational growth was discussed among the top managers in the group session at SupplyCo at length. The managers agreed that employing *too many* new engineers is not a good idea. According to a manager's opinion, recruitment efforts shall be proportional to the existing organization:

“You should aim for 33% impact [on the existing base]. For example, when 6 teams are trained, you can then add 2 teams every 2 months. When teams [complete the onboarding] then the new teams enter the preparation phase.”

And another manager explained:

“If we do not have a good competence breakdown figure and a large enough base, we cannot speed up the scaling. Newcomers dilute the existing teams, but the rest of the organization cannot help us a lot. We need more time to have the ability to integrate the people in the organization”.

There seem to be a threshold for reasonable organizational growth, unique for each organization and based on existing competence, resource availability and company size.

## 5 Concluding Discussion

In the previous sub-section, we described two stories of large onboarding campaigns. With respect to our research question: *What are the strategies for large onboarding campaigns?* We found that onboarding everyone at the same time is impractical and instead agile companies should organize large onboarding campaigns in batches (see Fig. 1). Further we know that effective onboarding of developers, especially in agile environment where detailed documentation is not emphasized, is organized through on-the-job onsite training [5, 14], however, in large-scale campaigns such a strategy is not possible. The often-used strategy further is to scale by splitting the recently onboarded teams (as shown in Fig. 2). However, the success of onboarding that puts engineers through these onboarding cycles is mixed. A deeper cross-case analysis suggests that the keys for successful large-scale onboarding are threefold.

- 1) **A good initial plan.** Companies shall have a good understanding of the complexity of the assignment and critically assess their own capabilities in terms of organizational maturity and size as well as existing in-house competences. A good onboarding plan for large campaigns includes a phase-based approach as suggested by related research [4, 5, 12] and a multiple-batch breakdown, a novel contribution of our study (see Fig. 2). Our findings shed light on the challenges and potential strategies for organizing multiple-batch onboarding, which resonate with the findings from an

unprecedented, transformational growth (although not as rapid as in our cases) reported by Kumar and Wallace [27]. Our results also suggest that the onboarding plan shall contain context-appropriate programs to avoid wasted training efforts. The helpful strategies included running self-study parts in parallel with on-the-job training (e.g., integration in the teams), which is consonant with existing research that emphasizes the importance of early productive work and exposure to work-related practices and ceremonies [5, 14]. Our findings contribute to existing research suggesting that mentoring and support can be organized through a “buddy team” or a “sister team” [6, 12, 14]. Prior research shows that such approach reduces the problems of mentors being unavailable for newcomers [14], and helps new teams to grow their social network, which is essential in large-scale agile [31]. And yet, our study clearly shows that following a plan will not take you far when changes arrive.

- 2) **Responsiveness to change.** Organizational agility, often seen as a key for success, is the ability of firms to sense environmental change and respond appropriately [30]. In our study, training targets shifted, recruitment got delayed, priorities of the ongoing operations changed, unforeseen gaps in documentation were uncovered and people started leaving. In both cases studied, supplier companies strived to learn from their experiences and respond to the emerging changes. Among the helpful strategies identified, many highlight the importance of constantly reevaluating the course of the onboarding through arranging daily synch meetings and retrospectives, as well as continuously improving the onboarding and training material. The importance of integrating the agile learning practices into onboarding are also presented in the model proposed by Gregory et al. [14]. Yet, our findings suggest that the ability to respond to changes requires certain organizational maturity.
- 3) **Organizational maturity.** Our study shows that large onboarding campaigns required certain organizational maturity and size to provide a safety net when the changes emerged. This is also highlighted by related studies from large and mature outsourcing suppliers who are able to address the challenges by relocating their key engineers to prioritized projects and customers [28, 29]. Our study thus suggests that large onboarding campaigns might not be for any company as the associated growth shall be proportional to the company size.

Evidently, the three keys to success generate further questions: How much planning should be built into the onboarding and scaling process? How much shall an organization invest into onsite training? How much scaling is appropriate for an organization of a certain size? Like the tale of the guitar player asking Buddha how best to tune his instrument, we find the famous answer, “Not too tight, and not too loose”. A better understanding of the thresholds for planning, investments into onsite training and organizational growth is a valid direction for future research in this area.

**Acknowledgements.** We would like to thank the studied client company for their engagement in our research, and especially for opening the doors to their outsourcing partners, which is uncommon in outsourcing research studies. This research is partially funded by the ScaleWise project and the Swedish Knowledge Foundation under the KK-Hög grant 2019/0087, and the Research Council of Norway through the 10xTeams project (grant 309344).

## References

1. Carmel, E., Tjia, P.: *Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce*. Cambridge University Press, Cambridge (2005)
2. Stray, V., Moe, N.B., Mikalsen, M., Hagen, E.: An empirical investigation of pull requests in partially distributed BizDevOps teams. In: *IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE)*, pp. 110–119 (2021)
3. Šmite, D., Moe, N.B., Gonzalez-Huerta, J.: Overcoming cultural barriers to being agile in distributed teams. *Inf. Softw. Technol.* **138**, 106612 (2021)
4. Bauer, T.N.: *Onboarding new employees: maximizing success*. SHRM Found. (2011)
5. Britto, R., Cruzes, D., Šmite, D., Šablis, A.: Onboarding software developers and teams in globally distributed projects: a multi-case study. *J. Softw. Evol. Process* **30**(4), e1921 (2018)
6. Buchanan, B.: Building organizational commitment: the socialization of managers in work organizations. *Adm. Sci. Q.* **19**(4), 533–546 (1974)
7. Klein, H.J., Polin, B., Leigh Sutton, K.: Specific onboarding practices for the socialization of new employees. *Int. J. Sel. Assess.* **23**(3), 263–283 (2015)
8. Lynch, K., Buckner-Hayden, G.: Reducing the new employee learning curve to improve productivity. *J. Healthc. Risk Manag.* **29**(3), 22–28 (2010)
9. Derven, M.: *Onboarding*, Association for talent and development (2008)
10. Gruman, J.A., Saks, A.M., Zweig, D.I.: Organizational socialization tactics and newcomer proactive behaviors: an integrative study. *J. Vocat. Behav.* **69**(1), 90–104 (2006)
11. Britto, R., Smite, D., Damm, L.-O.: Software architects in large-scale distributed projects: an Ericsson case. *IEEE Softw.* **33**(6), 48–55 (2016)
12. Moe, N.B., Stray, V., Goplen, M.R.: Studying onboarding in distributed software teams: a case study and guidelines. In: *Proceedings of the Evaluation and Assessment in Software Engineering*, pp. 150–159 (2020)
13. Sharma, G.G., Stol, K.J.: Exploring onboarding success, organizational fit, and turnover intention of software professionals. *J. Syst. Softw.* **1**(159), 110442 (2020)
14. Gregory, P., Strode, D.E., Sharp, H., Barroca, L.: An onboarding model for integrating newcomers into agile project teams. *Inf. Softw. Technol.* **143**, 106792 (2022)
15. Stray, V., Tkalic, A., Moe, N.B.: *The Agile Coach Role: Coaching for Agile Performance Impact* (2021)
16. Fagerholm, F., Sanchez Guinea, A., Borenstein, J., Munch, J.: Onboarding in open source projects. *IEEE Softw.* **31**(6), 54–61 (2014)
17. Maturro, G., Fontán, C., Raschetti, F.: Soft skills in scrum teams a survey of the most valued to have by product owners and scrum masters. In: *International Conference on Software Engineering and Knowledge Engineering*, pp. 42–45 (2015)
18. Steinmacher, I., Conte, T.U., Treude, C., Gerosa, M.A.: Overcoming open source project entry barriers with a portal for newcomers. In: *Proceedings of the 38th International Conference on Software Engineering*, pp. 273–284 (2016)
19. Morrison, E.W.: Newcomers' relationships: the role of social network ties during socialization. *Acad. Manag. J.* **45**(6), 1149–1160 (2002)
20. Casalnuovo, C., Vasilescu, B., Devanbu, P., Filkov, V.: Developer onboarding in GitHub: the role of prior social links and language experience. In: *ESEC/FSE*, pp. 817–828 (2015)
21. Boh, W., Slaughter, S.A., Espinosa, J.A.: Learning from experience in software development: a multilevel analysis. *Manage. Sci.* **53**(8), 1315–1331 (2007)
22. Šmite, D., Moe, N.B.: Switching vendors: factors that matter when engineers onboard own replacement. *J. Softw. Syst.* **169** (2020)

23. Yin, R.K.: Case Study Research and Applications: Design and Methods. SAGE Publications Inc. (2017)
24. Robson, C., McCartan, K.: Real World Research, 4th edn. Wiley (2016)
25. Creswell, J.W.: Qualitative Inquiry and Research Design, 3rd edn. SAGE Publications Inc. (2014)
26. Smite, D., Wohlin, C.: Strategies facilitating software product transfers. *IEEE Softw.* **28**(5), 60–66 (2010)
27. Kumar, S., Wallace, C.: Patterns of identity and interaction in an agile community of practice. In: Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering, pp. 71–78 (2019)
28. Moe, N.B., Šmite, D., Hanssen, G.K., Barney, H.: From offshore outsourcing to insourcing and partnerships: four failed outsourcing attempts. *Empir. Softw. Eng.* **19**(5), 1225–1258 (2013). <https://doi.org/10.1007/s10664-013-9272-x>
29. Sabherwal, R.: The evolution of coordination in outsourced software development projects: a comparison of client and vendor perspectives. *Inf. Organ.* **13**(3), 153–202 (2003)
30. Gren, L., Lenberg, P.: Agility is responsiveness to change: an essential definition. In: Proceedings of the Evaluation and Assessment in Software Engineering, pp. 348–353 (2020)
31. Moe, N.B., Olsson, H.H., Dingsøy, T.: Trends in large-scale agile development: a summary of the 4th workshop at XP2016. In: Proceedings of the Scientific Workshop Proceedings of XP conference, pp. 1–4 (2016)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# Organizational Conflicts in the Adoption of Continuous Software Engineering

Eriks Klotins<sup>1</sup>(✉) and Elliot Talbert-Goldstein<sup>2,3</sup>

<sup>1</sup> Software Engineering Research Lab (SERL), Blekinge Institute of Technology, Karlskrona, Sweden

`eriks.klotins@bth.se`

<sup>2</sup> Empirical and Applied Software Engineering Lab (EASEL), University of Maryland, Baltimore County, USA

`e13@umbc.edu`

<sup>3</sup> Bosserman Center for Conflict Resolution, Salisbury, USA

**Abstract.** Software is a critical component of nearly every product or service. Improvements in software can lead to substantial competitive advantages. At the same time, software and surrounding engineering teams have become increasingly complex.

The adoption of continuous integration and delivery is a recent trend to radically improve software release speed. However, its adoption is far from straightforward. Specifically, rethinking processes, organizational culture, ways of working, and business models require buy-in from diverse stakeholders that may have conflicting objectives. Such situations are explored by organizational conflict research.

This paper reports on early lessons from an ongoing research project in continuous software engineering, specifically investigating adoption challenges from an organizational conflict perspective. We identify catalysts, symptoms, and outcomes of organizational conflicts hindering the adoption process.

We conclude that predictable conflicts emerge when adopting continuous engineering. Engineers, managers, and other teams can proactively prepare for and allocate resources to resolve them. Proper analysis and management can help avoid wasted time, impeding processes, and frustration.

**Keywords:** Continuous Software Engineering · Organizational conflicts · Change management

## 1 Introduction

In this paper, we report on an ongoing research project into the adoption of continuous software engineering (CSE). Our focus is continuous integration and delivery, however we also consider broader organizational implications, such as planning and requirements on collaboration [6].

Continuous software engineering is a set of principles promoting rapid and frequent delivery of incremental software updates and extensive use of feedback to steer development [6]. Observing changes in product usage patterns due to

software changes informs engineers about the success of changes and support further product decisions [7, 11]. However, the key differentiating characteristic of CSE is the immediate delivery of any changes by developers to the end-users. This speed is achieved by extensive automation of build, test, integration, and delivery steps and by removing organizational bottlenecks.

Companies aiming to improve their software engineering processes by adopting continuous engineering principles face major challenges. Partly, the challenges are associated with adopting new tools and technologies. However, earlier studies suggest that the required organizational changes are the most significant challenge by far [2, 10, 11].

Deeper organizational inefficiencies are unlikely to be solved with new technologies and automation tools. Specifically, misunderstandings, misalignment, and sub-optimal organizational structures will limit the benefits of adopting process automation [6, 10].

Misalignment within and across organizations is addressed by organizational conflict research. Foundational research on organizational conflict and software engineering includes structural and interpersonal issues within and between organizations [1, 8, 12, 15, 21].

Studies of continuous integration and delivery primarily address social and technological obstacles. Such as challenges associated with team members changing roles and learning new skills and tools. However, only some, if any, focus on challenges establishing an efficient cross-organizational coordination and collaboration [3, 5, 13, 14]

We aim to understand to what extent underlying organizational conflicts hinder the adoption of CSE. We use three industrial cases to explore the adoption challenges and symptoms pointing toward deeper organizational inefficiencies.

Our results suggest that common adoption challenges, such as stakeholder resistance, restrictive processes, and organizational silos, can be explained by underlying organizational conflict. We conclude that organizations need to improve their conflict resolution approaches and advocate for continuous conflict management as part of the adoption strategy.

## 2 Background and Related Work

### 2.1 Organizational Conflicts in Software Engineering

In its most basic form, an organizational conflict is a misunderstanding or disagreement, real or perceived, around the needs, interests, and values of people working together [9].

CSE is a cross-cutting phenomenon requiring cooperation between all parts of an organization. The multitude of stakeholders and potential differences among them create particular causes of conflicts that need to be explored in the context of CSE [11, 16].

The causes of organizational conflict can be broken out into six broad categories: task interdependence, goal incompatibility, ambiguous rules, differences

in values and beliefs, resource scarcity, and ineffective communication [15]. These causes manifest differently in different scenarios, and specific components become more important for different types of conflicts.

Jehn [8] introduces an empirical typology of organizational conflicts. The topology covers the negative and positive impacts of conflict and incorporates both “management teams” and “production groups.” The three key types of organizational conflict are task, relationship, and process. Task conflict occurs when team members may disagree on how to perform work. The research indicates moderate task conflict can be constructive and stimulate discussion of ideas that help groups perform better and group productivity. However, relationship conflict, where teams have chronic issues that lead members to be “negative, irritable, suspicious, and resentful,” have significant adverse effects on productivity. Process conflict connects the two other types. Conflicts over *process* include how tasks should be accomplished, who is responsible, and how to delegate responsibility. Lower levels of process conflict were shown to limit the negative impacts on a team’s performance. This tripartite typology has become a foundation for analyzing team conflict, especially in software engineering.

Further studies have extended Jehn’s [8] work to include intergroup conflict [12, 21]. In software engineering specifically, this typology has been applied and empirically corroborated [9, 21, 22]. Enterprise software adoption research includes social network analysis to evaluate the impact of the propagation of intergroup conflict including issues with bureaucracy, interpersonal problems, and conflicting corporate values [21].

Siddique et al. [20] investigates the causes, consequences, and mitigation strategies of conflicts from the perspective of project managers. The causes include concerns like the role and knowledge of the product owner or customer, organizational hierarchy, bureaucracy, contracting and finances, and personal egos. Consequences included problems with productivity, wasted time and distractions, poor decisions, and lack of communication. The research covered possible strategies for addressing conflict, focusing on root-cause analysis as the primary means to resolve conflicts.

Power [17] introduces tools to study and remove impediments in Agile projects using impediment impact diagrams. Impediments included categories that comprised technical issues as well as interpersonal issues. The diagrams allow team members to compare the benefits of implementing a specific effort to remove an impact and the internal or external resources needed to do it.

Studying the types of conflicts that may occur during the adoption of CSE is not a direct route. It is essential to look at conflicts in teams and organizations generally and during organizational change. Additionally, it is helpful to narrow the scope of possible conflicts to software engineering and Agile contexts, which provide the foundations for CSE.

## 2.2 Continuous Software Engineering

Continuous software engineering originates from lean and agile principles. The overarching goal of CSE is to ensure a continuous flow of software from its inception to creating value for the user [6].

The continuous flow of software is achieved by delivering software in small increments and automating integration, verification and delivery steps. Frequent and small software deliveries creates an opportunity to gauge the success of the update by collecting focused telemetry and feedback [7, 11].

The benefits of CSE include the flexibility to react rapidly to new market opportunities and leverage data-driven decision-making. An automated pipeline allows engineers to focus on value-adding tasks and offload menial tasks to automation [6, 7]. The adoption process requires the organization to rethink its ways of working, set specific goals, and promote continuous improvements throughout the organization and its heterogeneous teams [10].

Implementing a CSE pipeline in an existing organization is a substantial undertaking. Adopting new tools, automation, test data preparation, and pipeline maintenance requires substantial investments. Furthermore, successful adoption of CSE requires cross-cutting changes in the organization [2, 11, 16]. For instance, to support the rapid delivery of software updates, the decisions of what to deliver need to be made rapidly. Fast planning has an upstream dependency on flexible resource planning and organizational support. Delivery of frequent updates requires buy-in from downstream stakeholders and customers. Such changes may require renegotiating customer contracts and tailoring business models [7, 11].

Our work with several industrial partners highlights that aligning the goals of different parts of an organization remains one of the key challenges in adopting continuous engineering. Misalignment often stems from different interpretations of the same goal. For example, improving efficiency may be interpreted by an R&D department as delivering more new features. For operations, efficiency may mean spending fewer resources on providing stable services. More frequent software updates increase the risk of disrupting smooth operations and are at odds with stability. In such situations, organizational conflicts may emerge and hinder attempts to improve organizational performance [10].

## 3 Research Methodology

The aim of our research is to explore how organizational conflict research can support organizations and practitioners in adopting CSE. To guide our study we define the following research questions:

**RQ1:** What are symptoms of organizational conflict in the context of adopting CSE?

*Motivation:* With this research question we want to explore to what extent difficulties in adopting CSE can be attributed to hidden and/or unaddressed organizational conflicts.



**RQ2:** What are the root causes of identified symptoms from an organizational conflict perspective?

*Motivation:* With this research question we want to explore the underlying causes for organizational conflict and map them to conflict resolution strategies.

**RQ3:** What are the advantages of studying conflicts in CSE empirically?

*Motivation:* Most of research focuses on tooling and technical aspects of CI/CD. However, a degree of organizational streamlining are required to enable end-to-end automation. With this research question we aim to clarify how studying organizational conflict can support adoption of CSE.

### 3.1 Research Approach

We conduct this study as part of an ongoing industry-academia research project into adopting continuous software engineering in the industry. The project aims to develop support for practitioners to adopt and benefit from CSE principles.

As part of the project, we conduct a multiple case study [18]. The studied cases are established organizations with mature products already in the market. The unit of analysis is the current software delivery process in developing market-driven software-intensive products. Our primary data collection methods are interviews, workshops, and seminars. Our analysis focused on the process, identification of bottlenecks, and opportunities to improve the efficiency and effectiveness of the software delivery process.

This paper is based on results from three partner organizations. We name companies A, B, and C to maintain their anonymity. The work with these organizations was conducted between January 2022 - February 2023.

*Company A* is a large mobile telecommunications hardware and software provider in Sweden. We were involved with a part of the organization developing business-critical software systems for mobile telecommunications operators. Their software release process is based on a quarterly release cycle. To achieve exceptional quality and compliance, the release process is concluded by a sign-off stage involving many stakeholders representing trade compliance, security, business, engineering, and customer representatives, among others.

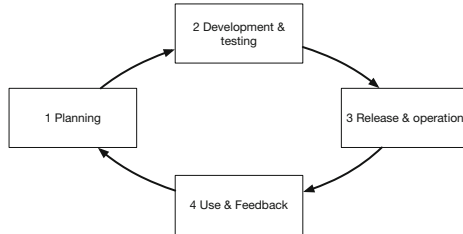
*Company B* is an audio streaming services company based in Sweden. Their offerings include mobile apps, services, and tools to bring musicians, audio content creators, and listeners together. Their organization is relatively new and characterized by a flat structure, agility, and flexibility. Teams can release software changes within hours. However, coordinating large and cross-cutting efforts is challenging.

*Company C* is a mobile telecommunications operator in Scandinavia. Their offerings consist of both consumer and business oriented mobile telecommunications services. The organization also develops tools for internal business support. Internal efficiency and speed of getting software changes from an idea to production are their major concerns.

## 4 Results

The first interview rounds focused on understanding how software development and delivery process works and what known bottlenecks are. Later interview and workshop rounds were focused on specific bottlenecks.

The interviews quickly revealed that all three organizations had already implemented some degree of automation and attempted to streamline their engineering processes. For instance, test, build, and integration automation was successfully adopted and led to substantial perceived improvements. However, the automation of development tasks is isolated in the development and testing stage of the process, see Fig. 1.



**Fig. 1.** Overview of the key steps of CSE

Further interviews with companies A and C revealed that significant bottlenecks to rapidly delivering changes occur after development is considered completed. That is, preparing and handing over completed software for release and operations, see Fig. 1. Namely, the handover includes preparing documentation to ensure knowledge transfer, seeking sign-offs from multiple stakeholders, and verifying regulatory compliance, among other activities.

All our partner companies mentioned the intention to use product telemetry and customer feedback to inform further product planning, see Steps 1 and 4 in Fig. 1. However, access to customer environments and data can be problematic due to security and privacy concerns, service level agreements, business risks, lack of trust, or inadequate tooling to manage large volumes of data. Furthermore, due to organizational silos and the lack of transparency, the existing data may not be available to other stakeholders.

Further interviews and workshops revealed that the main obstacles to smooth and continuous software delivery are misinterpreting organizational objectives, leading to organizational silos and stakeholders protecting their interests. This inter-team conflict manifested largely as issues with processes, and teams had difficulty overcoming them. Conflicts like this are not unique to CSE. However, some specific issues arise in understanding and addressing them that need to be studied so that future organizations can address them more easily. It is possible to address these conflicts through analysis and mitigation efforts to improve the performance of CSE efforts.

Further insights from work with companies led us to identify catalysts, symptoms, and outcomes associated with adopting continuous practices and unresolved organizational conflicts.

#### 4.1 Catalysts of Organizational Conflict Hindering the Adoption of CSE Principles

Catalysts do not necessarily lead to conflict or other issues, however, they are mentioned as a context where issues are more likely to emerge.

*Catalyst-I: Functional organizational structures* emerge from how the organization is built. Our interviewees pointed out that companies are often structured around R&D, operations, sales, product planning, strategic management, and customer support functions. In such organizations, each unit has a distinct function and objectives to fulfill. Any improvements are often limited to each function without considering a broader picture. A consequence of such structures and local optimizations are organizational silos [19].

Companies A and C pointed out that a cross-cutting end-to-end software delivery, see Fig. 1, requires the collaboration of multiple organizational functions. Communication between the silos is often tricky due to a lack of context, mismatching goals, and different specializations. Slow release cycles contribute to lost knowledge and context. Due to issues like artificial boundaries, unspecified goal alignment, or lack of oversight into potential issues, there is an increased risk of conflict between the teams. As one interviewee from *Company A* put it:

*“R&D teams have no idea of how their work is integrated and delivered to customers. We often discover many issues late in the process and we have to go back [to development] or try to find solutions on-the-fly.”*

In *Company B*, the primary organizational unit is a cross-functional team. Each team comprises a product manager, engineers, infrastructure and delivery specialists, data scientists, and other roles. Such a structure ensures that the complete planning, development, delivery, and feedback collection cycle, see Fig. 1, can be executed within one team. This structure keeps communication distances small and minimizes cross-unit communication and coordination.

*Catalyst-II: Inimical processes* lead to sub-optimal organizational performance due to lack of flexibility. We observe that organizations have a tendency to grow their processes by adding more steps and including new stakeholders. Processes become inflated due to over-complicated procedures with many stakeholders who do not necessarily understand their role in the process or share the same goals. At the same time, our industry partners report the need to become faster and offer new types of software deliveries. Our interviewees shared their interest in simplifying processes by removing activities and stakeholders. As one interviewee put it:

*“Our [inimical] process did not help us to catch the log4j issue. It only delayed us in releasing the fix.”*

Company-A pointed out that a process built for steady delivery of business-critical software hinders rapid delivery of security updates or experimental features. When the process and the involved stakeholders do not permit enough room for flexibility, it leads to frustration and sub-optimal performance. Moreover, frustration leads stakeholders to seek shortcuts and use their influence in the organization to get things done. Importantly, processes involving ad-hoc negotiations between stakeholders are difficult to automate and streamline.

*Catalyst-III: Gatekeeping managers* have a role in halting a process until certain conditions are met. They are often part of inimical processes (Catalyst-II). For instance, a gatekeeper would be a senior manager signing off a release for delivery to customers, a council scrutinizing reports to negotiate if the software meets a quality threshold, and alike. While some level of gatekeeping is necessary, a challenge arises when the gatekeeper lacks contextual knowledge and in-depth understanding of the artifacts produced by the process.

As reported by Company A, the gatekeepers are often responsible for signing-off many different artifacts. They have a limited capacity to have an in-depth understanding of what they are signing off on. Getting the gatekeeper's knowledge to a sufficient level for an informed decision takes time and is impractical, given the size and complexity of the software. Slow release cycles and parallel processes lead to the lost context that needs refreshing every time. Hence, the gatekeeper is unlikely to fulfill its function and hinder the pace of software deliveries. As one interviewee put it:

*“To get an approval, I need to book a meeting with 20 stakeholders. I send them information upfront, but they do not read it in many cases. This repeats every six months”*

In Company B, the gatekeepers are primarily within the team responsible for conducting the work. Thus, they have full contextual knowledge and understanding of the artifacts to make an informed decision.

Based on company comments, it did not appear that the experiences associated with gatekeeping were amicably resolved. Instead, process conflict defined the interaction, where no side could agree on particular operations. Team members who disagree with a set process are likely to become frustrated, and this conflict can impact overall performance and lead to further conflicts between people.

*Catalyst-IV: Lack of Autonomy and Support for Conflict Resolution.* The above catalysts are exacerbated by the lack of autonomy and support for resolving conflicts. Companies A, B, and C follow agile software engineering practices and empower their engineers to make decisions to a certain degree. However, we observe that organizational conflicts are likely to occur during the handover of artifacts from one organizational unit to another.

Different organizational units may follow different principles and fall under different managers. Thus, resolving any disputes may require an agreement between the parties that they have the authority to resolve the conflict or requires

taking several steps up the management chain. As illustrated by Company A, different organizational units often have different objectives and are physically located on different continents. The negotiations between organizational units happen, however, they are repeated at every software delivery cycle, causing delays and frustration. The involved parties need the authority to adjust their ways of working. Companies B and C are smaller, and engineers have more authority to adjust their ways of working. As *Company B* put it:

*“We have the authority and the responsibility to do the right thing.”*

In particular, team members, and project managers need to be adequately empowered and trained to resolve interpersonal conflict. There will be negative impacts. In CSE efforts, the team manager may be able to negotiate over particular deliverables and tasks, but issues with processes are less likely to be managed effectively. Additionally, process conflicts are more likely to propagate to other areas than lower-level task issues, leading to broader negative impacts.

## 4.2 Symptoms of Underlying Conflicts

Process analysis in Companies A, B, and C revealed a number of challenges hindering rapid software delivery. In our study, these challenges emerged as symptoms of deeper challenges.

Notably, the adverse effects emerge as an overhead for every software delivery cycle. Increasing the pace of software deliveries without optimizing the overhead would compound the overhead and waste.

In our study, we identify the following symptoms of underlying organizational conflicts:

*Symptom-A: Repeated Negotiation Every Time an Activity or Process is Performed.* All partners mentioned that they depend on meetings to coordinate work. However, interviewees from Company A described that they often have meetings to renegotiate certain decisions every time a process is performed. Such meetings take calendar time to schedule and require informing the stakeholders about the decision context, and in many cases, add little value because the outcome of the decision is expected to stay the same. Our interviewees mentioned that they are used to a release sign-off meeting with about 20 stakeholders formally approving a release. Such a process is acceptable with quarterly releases, however, is impractical when the organization aims to achieve faster release cycles.

Due to inimical processes (Catalyst-III) and the lack of autonomy (Catalyst-IV), stakeholders cannot skip unnecessary negotiations. An individual project or team member does not have the authority to change the process. Should they attempt to do so, more conflict could erupt. In this scenario, the question becomes “who is responsible for process improvement?” or “how do we achieve trust in faster release cycles?.” A party such as a senior manager or project sponsor must be empowered to arbitrate or oversee the removal of processes that act as impediments for teams to change outdated procedures while maintaining the autonomy of teams.

*Symptom-B: Organizational Constraints Hindering Effective Work.* The organizational constraints arise from organizational structures, internal policies, lack of transparency, and the lack of efficient mechanisms to adjust the constraints. In its most severe case, organizational constraints manifest as competing silos.

Company B's main organizational unit is a small, autonomous, cross-functional team. Such a structure makes the organization very efficient in delivering work on a team level. However, coordinating the work of many autonomous teams to deliver more extensive work is challenging. The challenge arises from juggling different priorities on different levels. For instance, a team may be required to support multiple other teams while delivering on their objectives.

Company A has a functional structure, and organizational units have different objectives and specializations. For instance, the responsibility to deliver working software to customers is delegated to one final integration team. Other teams provide software components and have a partial picture of how their components will be integrated and used. Thus, they cannot make informed decisions about their work.

A shared concern in Companies A and B is access to product and process data. The process data is required to enable transparency of other teams' progress and efficient work planning. Product usage data is required to infer customer preferences and steer product development work. However, due to data protection concerns, lack of tooling, and ad-hoc data collection methods, such data is not freely available throughout the organizations.

Company B has an internal team and processes responsible for aggregating data from various sources. However, due to differences in how teams plan their work, there needs to be oversight of the current work progress and resource availability. Company A has an overview of the work progress due to the central planning function, however the data of how software is used is not shared across organizational functions.

*Symptom-C: Blocking Task Dependencies.* Task dependency arises from the need for multiple organizational units to collaborate and coordinate toward a common goal. Challenges arise when one unit needs to wait until another completes its part. Or when there is a gap between what one unit delivers and what another expects.

In Company A, R&D task dependencies are minimized and well managed. However, difficulties arise in coordination with software delivery, operations, and sales units. Due to the complexity of software, bespoke versions delivered to customers, and lack of transparency, the deliveries do not always match the expectations. Filling the gaps requires substantial effort just before releasing software to customers.

In Company B, teams can collaborate in different constellations. Coordinating and scheduling a large volume of tasks is a major challenge. The challenge can be partly attributed to the lack of company wide task and resources tracking system.

*Symptom-D: Utilization of Personal Contacts and Process Shortcuts.* Inimical processes (Catalyst-II) and difficulties in reaching consensus push stakeholders to use their influence, networks, and process loopholes to accomplish their objectives.

Several interviewees mentioned using personal relationships to reach out to other stakeholders and influence them to behave in a certain way. We observe that such behavior is more prevalent among more senior employees. At the same time, more junior interviewees mentioned the difficulties of attaining the same outcome through formal channels.

Another manifestation of this symptom is labeling artifacts in a certain way to simplify the process. For instance, an interviewee mentioned labeling a major update as a hot fix to avoid a lengthy, and in their view, pointless review process.

### 4.3 Outcomes from Unresolved Organizational Conflicts

Our partner companies report a number of negative outcomes associated with the symptoms. Importantly, the outcomes become significant concerns when organizations plan to speed up their release cycles and improve internal efficiency and effectiveness.

*Outcome-A:* Wasted time. Leaping organizational boundaries, renegotiating the same decisions, involving distant stakeholders in making critical decisions takes time away from more value creating activities. The time is wasted every time a process runs.

*Outcome-B:* Lack of flexibility. Inimical processes, dependency of personal networks to get things done, and lack of autonomy to resolve inefficiencies limits the organizational ability to adapt to new market and business conditions.

*Outcome-C:* Frustration. Awareness of organizational inefficiencies lead to frustration among stakeholders. In turn, frustration leads to reduced motivation, increased turnover rate, and overall reduced organizational performance.

## 5 Discussion and Analysis

This research collected information broadly about adopting CSE in large companies with multiple software products. The focus of the data collection was not explicitly aimed at the conflict. Nevertheless, the insights touched upon conflicts throughout the process and bore further investigation.

Our results show that adopting CSE is as much an organizational as it is an engineering challenge. Thus, a study into CSE adoption must balance the technical and social lines of inquiry.

Studying the organizational impediments to CSE adoption falls squarely in the realm of the study of people and organizations, which managers must be familiar with to be successful. The company's experiences in these cases revealed conflicts that organizations are likely to face when implementing CSE and Agile methodologies, more broadly.

### 5.1 RQ1: What are Symptoms of Organizational Conflict in the Context of Adopting CSE?

There are a number of organizational concerns to be addressed when adopting CSE practices. We separate these concerns into catalysts and symptoms.

The catalysts are not problematic or cause conflict per se. However, our results suggest that functional organizational structures (Catalyst-I), inimical, die-hard processes (Catalyst-II), the culture of sign-offs (Catalyst-III), and the lack of autonomy and support for conflict resolution (Catalyst-IV) create conditions for conflicts to emerge.

Symptoms are signs of deeper issues. We identify that the culture of repeated meetings and decisions with the same outcome (Symptom-A), organizational “red tape” (Symptom-B), blocking task dependencies (Symptom-C), and the utilization of process shortcuts (Symptom-D) signal a deeper conflict.

These results are well aligned with the state-of-the-art in CSE. For instance, cross-functional teams, empowering engineers to decide on their ways of working, and organizational transparency are prerequisites for the successful adoption of CSE [7]. At the same time, the existing literature does not provide a deeper insight on the organizational change management perspective [10,15]

### 5.2 RQ2: What are the Root Causes of Identified Symptoms from an Organizational Conflict Perspective?

State-of-the-art management practices recognize several groups of factors contributing to organizational conflict. These categories include issues caused by task interdependence, goal incompatibility, ambiguous rules, differences in values and beliefs, resource scarcity, and ineffective communication [15].

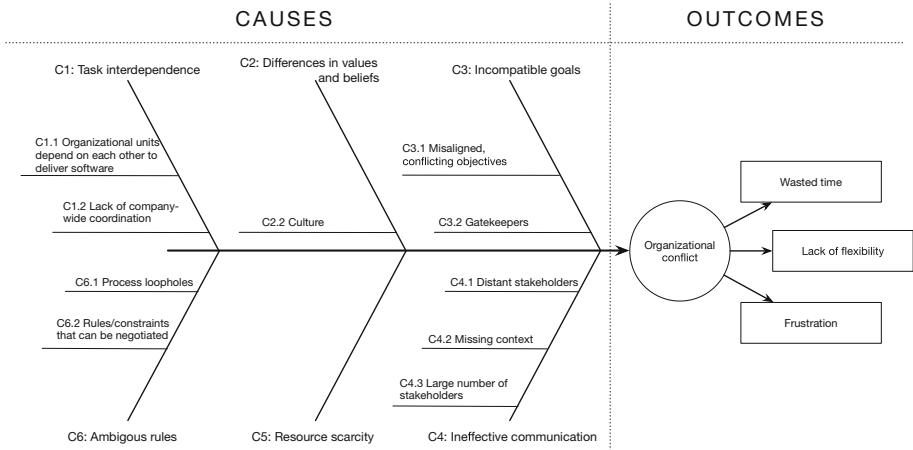
We use the causes of organizational conflict to analyze insights from our study. Under each cause for the organizational conflict, we list sub-causes mentioned by our interviewees, see Fig. 2.

Our observations match well with the state-of-the-art in organizational conflicts. However, conflict management has attracted little attention from CSE community. Moreover, discussions with our partners reveal that the adoption of CSE is treated mostly as an engineering problem with the focus on technical aspects. Our findings demonstrate how including conflict management in the CSE adoption strategy could alleviate some of the adoption challenges [16].

Software engineering and delivery of software-intensive products and services are inherently collaborative. Thus, some degree of dependency and conflict is inevitable, see C1.1 in Fig. 2. In Company A, cross-functional dependencies are more common. However, in Company B, the team dependencies are more common, thus conflicts occur on different levels.

Our interviews confirm that collaboration across organizational units indeed causes friction, see C1.2 in Fig. 2. From Company A, we learn that well-coordinated work in the R&D unit can solve many issues. At the same time, the coordination does not extend beyond the functional R&D unit. This is a potential cause of conflict.





**Fig. 2.** Root-cause analysis of organizational conflicts in adopting continuous software engineering.

Topics around poorly understood and misaligned organizational objectives were discussed during our interviews, see C3.1, in Fig. 2. We conclude from the interviews that setting multiple organization-wide goals with shared and clear key performance indicators is challenging.

In Company A, the primary goal for a long time had been the quality of services. Consequently, there is a rigorous quality assurance process. Recently, the organization sought to increase delivery speed. This move requires rethinking and streamlining the quality process. In particular, the gatekeepers, see C3.2., need to be replaced with an automated and scalable solution. At the same time, the priorities of quality and speed are not shared, and some fears removing the established process may lead to deteriorating product quality.

In Company B, the primary goal is delivery speed. The organization had been optimized to deliver features to the market fast. At the same time, the ability to track the market relevance of high-level updates is limited.

We observe that many stakeholders across organizational units cannot maintain the full context of the situation, coordinate and make efficient decisions. Consequently, fault lines appear, and organizational conflicts emerge.

This challenge is most prevalent in Company A, where the key stakeholders have a many-to-many mapping with the ongoing projects with long release cycles. Thus, they cannot fully understand each project's context.

The causes and effects of the conflicts we identified are directly related to the shift to CSE. These issues may not arise in other business transformations and require special attention in this context. How these conflicts manifest for different companies differs and depends on the legacy structures and processes, as well as overarching culture and goals. Team members can be conscious of these prospective issues and flag them should they appear. Managers can also be pre-

pared to address them based on best practices in order to maintain performance and meet goals.

Initially, managers can take stock of the catalysts we identified here. These represent the corporate structures that can lead to conflicts later on. Where possible, leaders should prepare for CSE adoption by restructuring some of the organization, or its processes at least, to address these root causes. At a minimum, engineering teams should be prepared to deal with the symptoms that arise from these catalysts. Failing that, (for example, if you read this research after starting a CSE change) team members should be aware of the symptoms that are driven by the catalysts. If these symptoms arise, they can likely be traced back to their root causes at the organizational level to seek solutions. Performing a root cause analysis on symptoms not defined here could also reveal new catalysts.

### **5.3 RQ3: What are the Advantages of Studying Conflicts in CSE Empirically?**

Studying the conflicts in CSE, including their symptoms and catalysts helps prevent them, and also helps too maximize their positive outcomes when conflicts inevitably occur. For instance, since some degree conflict in teams is healthy [8], managers should reflect on them with their teams during reviews and retrospectives, to celebrate the wins and encourage a positive discord. Embracing positive conflict could be a driver for incremental organizational change.

Tracking the frequency of conflicts is also valuable. A lack of conflict may indicate instability in a team or organization, since stakeholders may avoid raising disputes for fear of causing their group to crumble. Instead, observing a manageable amount of healthy conflicts can help indicate team cohesiveness [4]. Managers should clue-in to task conflict that results in positive outcomes, and raise a red flag when teams grow silent.

Lastly, the goals of adopting CSE are meant to close the gaps between the business need for a software solution and its development, without negative impacts to quality in the short- or long-term [7]. As with any Agile transformation, and for any project manager, removing obstacles is the key priority. For CSE, those obstacles have been shown to surface as issues between people more than from technical limitations. Just like change is inevitable, so too is conflict. Organizations should thus prioritize management, and to some extent welcome, disputes that arise during adoption of CSE, particularly between heterogeneous organizational units. This means, for example, facilitating dialogue to focus disparate interests towards shared goals.

By systematically identifying, addressing conflicts that naturally occur, organizations can help ensure the success of a continuous pipeline of software that meets the needs of customers, engineers, operations, and sales.

## 6 Conclusions and Further Work

In this paper, we report preliminary results from an ongoing study on adopting CSE in three companies. Through interviews and workshops, we observed challenges faced by different teams within each organization. The data regularly included participant discussion of organizational conflicts, defined by differences in the understanding of goals, needs, and interests between teams. Based on this empirical information, we have identified catalysts, symptoms, and outcomes associated with improperly managed organizational conflict that inhibits the adoption of CSE. Unless it is addressed, organizational conflict can hinder the implementation of a continuous software delivery pipeline and the surrounding automation.

We conclude in this early work that analyzing organizational conflict, and managing it constructively, are important parts of streamlining an organization and realizing the benefits of CSE. We propose to recognize *continuous conflict management* as an essential and cross-cutting activity in the toolbox of implementing continuous integration and continuous delivery.

Further work should look to these underlying conflicts, their causes, and their outcomes, as a starting point to identify further conflicts that may arise. Quantitative studies could also be used to measure the frequency of such conflicts and evaluate their impacts.

## References

1. Afzalur Rahim, M.: Toward a theory of managing organizational conflict. *Int. J. Conflict Manag.* **13**(3), 206–235 (2002)
2. Chen, L.: Continuous delivery: overcoming adoption challenges. *J. Syst. Softw.* **128**, 72–86 (2017)
3. Claps, G.G., Svensson, R.B., Aurum, A.: On the journey to continuous deployment: technical and social challenges along the way. *Inf. Softw. Technol.* **57**, 21–31 (2015)
4. Coser, L.A.: *The Functions of Social Conflict*, vol. 9. Routledge (1998)
5. Debbiche, A., Dienér, M., Berntsson Svensson, R.: Challenges when adopting continuous integration: a case study. In: Jedlitschka, A., Kuvaja, P., Kuhrmann, M., Männistö, T., Münch, J., Raatikainen, M. (eds.) PROFES 2014. LNCS, vol. 8892, pp. 17–32. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-13835-0\\_2](https://doi.org/10.1007/978-3-319-13835-0_2)
6. Fitzgerald, B., Stol, K.-J.: Continuous software engineering and beyond: trends and challenges. In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, pp. 1–9 (2014)
7. Humble, J., Kim, G.: *Accelerate: the science of lean software and devops: building and scaling high performing technology organizations*. IT Revolution (2018)
8. Jehn, K.A.: A qualitative analysis of conflict types and dimensions in organizational groups. *Adm. Sci. Q.* 530–557 (1997)
9. Karn, J.S., Cowling, A.J.: Measuring the effect of conflict on software engineering teams. *Behav. Res. Methods* **40**, 582–589 (2008)
10. Klotins, E., Gorschek, T.: Continuous software engineering in the wild. In: Mendez, D., Wimmer, M., Winkler, D., Biff, S., Bergsmann, J. (eds.) SWQD 2022. LNBIP, vol. 439, pp. 3–12. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-04115-0\\_1](https://doi.org/10.1007/978-3-031-04115-0_1)

11. Klotins, E., Gorschek, T., Sundelin, K., Falk, E.: Towards cost-benefit evaluation for continuous software engineering activities. *Empir. Softw. Eng.* **27**(6), 157 (2022)
12. Korsgaard, M.A., Soyoung Jeong, S., Mahony, D.M., Pitariu, A.H.: A multilevel view of intragroup conflict. *J. Manag.* **34**(6), 1222–1252 (2008)
13. Laukkanen, E., Itkonen, J., Lassenius, C.: Problems, causes and solutions when adopting continuous delivery—a systematic literature review. *Inf. Softw. Technol.* **82**, 55–79 (2017)
14. Laukkanen, E., Paasivaara, M., Arvonen, T.: Stakeholder perceptions of the adoption of continuous integration—a case study. In: 2015 Agile Conference, pp. 11–20. IEEE (2015)
15. Mitchell, D.E.: *Causes of Organizational Conflict*. Springer, Cham (2017)
16. Neely, S., Stolt, S.: Continuous delivery? easy! just change everything (well, maybe it is not that easy). In: 2013 Agile Conference, pp. 121–128. IEEE (2013)
17. Power, K.: Impediment impact diagrams: understanding the impact of impediments in agile teams and organizations. In: 2014 Agile Conference, pp. 41–51. IEEE (2014)
18. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. *Empir. Softw. Eng.* **14**, 131–164 (2009)
19. Serrat, O.: Bridging organizational silos. In: *Knowledge Solutions: Tools, Methods, and Approaches to Drive Organizational Performance*, pp. 711–716 (2017)
20. Siddique, L., Hussein, B.A.: Grounded theory study of conflicts in Norwegian agile software projects: the project managers’ perspective. *J. Eng. Project Prod. Manag.* **2**, 120–135 (2016)
21. Williams, R.A.: Conflict propagation within large technology and software engineering programmes: a multi-partner enterprise system implementation as case study. *IEEE Access* **7**, 167696–167713 (2019)
22. Zhang, X., Stafford, T.F., Hu, T., Dai, H.: Measuring task conflict and person conflict in software testing. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **29**(4), 1–19 (2020)


**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.





# Data-Driven Development in Public Sector: How Agile Product Teams Maneuver Data Privacy Regulations

Astri Barbala<sup>1</sup> , Tor Sporsem<sup>1</sup> , and Viktoria Stray<sup>1,2</sup> 

<sup>1</sup> SINTEF Digital, 7034 Trondheim, Norway

{astri.barbala, tor.sporsem, viktorija.stray}@sintef.no

<sup>2</sup> University of Oslo, 0373 Oslo, Norway

**Abstract.** Datafication processes, the ongoing strive for making organizations data-driven, have in recent years entailed data-focused software projects and more interdisciplinary teamwork. Simultaneously as agile product teams have been directed towards increased use of data for software development, stronger data protection regulations such as GDPR have further complexified the software developer role, whose responsibilities and expectations now expand far beyond mere coding. Seeking to develop an understanding of how data-intensive product teams in the public sector maneuver the legal hurdles emerging in the wake of data governance, this paper builds on 19 interviews with members of two agile product teams in the Norwegian organizations NAV and Entur. Our findings indicate that including a legal expert in the team can boost confidence in data handling practices and avoid delays in deliveries, but it requires effort to synchronize and overcome interdisciplinary barriers.

**Keywords:** Datafication · Data Protection · Data Management Regulations · GDPR Compliance · Agile Software Development · Collaboration · Coordination · Interdisciplinary Teams

## 1 Introduction

Over the last decade, European governments have looked to data-driven technology to enable “the strategic use of data for productive, inclusive and trustworthy governance” [1], meaning that data collection, storage, and sharing are playing increasingly central roles in public administration. The quest for a datafied public sector is largely driven by a vision of both empowered citizens and better and more effective public services, entailing reliable and secure software solutions built with agile methods. However, research has shown that there are several multifaceted constraints involved in becoming data-driven, both technical, organizational, political, and legal (see, e.g., [2–4]).

For software developers, the direct implication of datafication is that product teams must own and manage data as part of their everyday practice, resulting from data mesh structures and the move towards decentralized data storage [5] and democratization of data [6]. Simultaneously, as data science has rapidly made its way into software

development, agile public sector teams are expected to maneuver new team constellations and often ambiguous data management regulations.

A central issue for public sector organizations in the quest of becoming data-driven entails overcoming juridical hurdles effectively and in accordance with applicable laws, specifically to protect citizens' private data. This paper seeks to gain an understanding of how data-intensive public sector product teams can overcome these hurdles and how scenarios can play out differently depending on the team composition; more specifically whether the team includes a legal expert or not. It is well-established that the barriers to the effectiveness of agile teams lie not only at the team level and leadership of the team but also in the organizational and environmental contexts that directly affect team success [7]. However, few empirical studies have to date zoomed in on how data privacy laws affect the day-to-day work of software developers and the importance of team composition and organization in that regard.

To address this gap in the literature, we studied two agile product teams in two central public sector organizations in Norway, NAV and Entur, that were transitioning from centralized to distributed data management. "Product teams" are autonomous teams with the cross-functional competence needed to provide the service or product independently<sup>1</sup>. The two agile teams develop data-intensive products, meaning utilizing user data in the software development process, and both software developers and team members from other disciplines were interviewed. We chose to explore the following research questions:

- 1) How are data privacy regulations affecting the day-to-day work of product teams?
- 2) What are the pros and cons of including a legal advisor as part of the team to overcome juridical hurdles?

This paper is organized as follows: In Sect. 2, we introduce public sector datafication and what it entails in the context of agile software engineering. We then explain the increasing importance of juridical understandings for agile product teams, underlining the direct impact data privacy regulations have on present-day software development. Section 3 describes the context and methods before we present our findings in Sect. 4. In Sect. 5, we discuss the main data privacy challenges for agile product teams and discuss how these obstacles can be addressed moving forward. Section 6 concludes and suggests future work.

## 2 Background

### 2.1 Defining Public Sector Datafication: Towards Decentralized Data Management

Data has been viewed as the new oil, entailing that enormous value can be extracted from refined data and that just like oil spills, "data spills" can cause tremendous damage [8]. Within software engineering, datafication has also been regarded as an emerging

---

<sup>1</sup> Both companies are heavily inspired by the Team Topologies book and use "product teams" as synonyms to "stream-aligned teams". See Skelton, M., & Pais, M. (2019). *Team topologies: organizing business and technology teams for fast flow*. It Revolution.

“trending topic” and an aspect of the field’s increasing focus on data [9–11]. Datafication can be defined as “the transformation of social action into online quantified data” [12], and entails “the collective tools, technologies, and processes used to transform an organization to a data-driven enterprise” [13].

Although much research has been conducted on public sector datafication in the last few years in terms of policy implications and citizen rights and participation [4, 14–16], there has to date been a lack of scholarly interest within the software engineering field for discussing how datafication has affected software projects and the software developer role. Systematic data about populations have been collected in the Nordic countries for a long time, partly due to the development of welfare infrastructures [17]. Technology researcher Jathan Sadowski [18] has stated that “data – and the accumulation of data – is a core component of political economy in the 21st century”. Data is thus a potential source of citizen insight and enormous capital and entails that public sector product teams can be defined as particularly data-intensive [19].

In developing digital public services, such as online applications for filing tax returns and unemployment benefits, the Norwegian public sector has increasingly turned towards distributed data management models that are arguably more compatible with modern-day agile software development [5]. Data mesh is defined by Zhamak Dehghani [20, 21] as fulfilling four principles; namely 1) domain-oriented decentralized data ownership and architecture, 2) seeing data as a product, 3) incorporating a self-serve data platform and 4) a turn towards federated computational governance. The direct implication of data mesh for software practitioners is that product teams gain more control and ownership of the data deriving from the team’s designated application, in addition to increased responsibility to share data with other teams. NAV and Entur have been public sector pioneers in turning towards data mesh incorporation, which incentivized us to choose teams from these organizations as our focus points for this study.

## 2.2 The Legal Aspect of Agile Teams: An Increasing Importance

In recent years, with the advent of data privacy regulations governing how best to collect, store and utilize the value of data, maneuvering the oftentimes ambiguous and little context-specific data management rules has become an increasing challenge for both public and private sector software teams. User-centered laws such as the General Data Protection Regulation (GDPR) launched by the European Union in 2018 have directly affected European software development planning and implementation. As a member of the European Economic Area (EEA), Norway is bound to the GDPR. Additionally, Norwegian law has national adaptations that are tailored more directly to the country [22].

The new and stronger rules on data protection entail that people have more control over their personal data, as well as giving businesses and organizations several obligations in terms of carrying out data protection assessments and maintaining records. These new responsibilities have led software development teams to look outside IT-related professions to fill the emerging roles to become self-organizing [23]. Integrating experts from other disciplines into IT teams to strengthen their agility and interdisciplinary competence is nothing new. However, it has become increasingly important in recent years with more and more complex software projects and team compositions further complicated by data science roles [24]. In addition, the turn to continuous software

deployment means that teams are increasingly expected to deliver faster and faster, going from a couple of releases a year to several a day.

Ensuring compliance with relevant regulations is a vital component of the continuous integration strategy. As pointed out by Klotins et al. [25], regulatory practices are often at odds with agile and continuous principles. Previous research has shown that adhering to GDPR is burdensome for software developers due to both lack of support from institutions and clients, and inadequate online tools [26]. Although continuous compliance is an established procedure within agile literature, there is a lack of empirical studies into how this plays out in practice in data-driven software teams. This study thus seeks to add new insights into how the compliance principle may create congestion in the continuous delivery process due to a limited understanding of data usage legalities.

The importance of juridical competence for software teams has meant that IT departments have started implementing legal professionals into product teams, aiding software developers in maneuvering the legal grey zones in datafication processes. Our present study contrasts one team with another team that did not have a legal advisor available, and we hence seek to provide unique insights in this regard. Although organizational agility is critical to respond sufficiently to the challenges experienced by contemporary software projects, team members with different backgrounds may have different norms guiding them, often proving to be a hindrance to being an effective agile team [27]. Addressing this in the context of a product team incorporating a legal advisor, we also offer novel empirical findings contributing to this literature in software engineering, as this aspect is not discussed in any of the previous literature as presented in the present section. Additionally, the paper adds to the state of the art in that the Norwegian public sector (and especially NAV and Entur) has been a global pioneer regarding incorporating new ideas about organizational architecture in software development, to date still largely untheorized within empirical studies of public sector software engineering.

### 3 Method and Research Site

To answer the research questions, we utilized a qualitative case study design [28] to tell the contrasting stories of two agile teams in NAV (Team Welfare) and Entur (Team Travel). We chose these cases because both are public companies with a data mesh strategy for becoming data-driven (i.e., decentralizing data ownership to product teams), allowing us to compare two cases within similar contexts. Team Welfare included a legal advisor, which Team Travel did not. We wanted to explore differences in overcoming privacy issues in the two teams and the collaboration between developers and team members with legal expertise. Our goal was to shed light on the challenges faced in the pursuit of becoming a data-driven public sector. Specifically, how to establish and maintain legal standards and procedures for effective data handling.

We kept an exploratory approach as we did not set out to test any specific theory or hypothesis [29]. Further, we hold an interpretive view in this study, comprehending the world and its truths as subjective realities [30]. As per Yin's [28] approach to case study research, case studies are not intended to be statistically generalized, and this can thus be said to be a limitation of case studies and qualitative studies in general. However, the study can be analytically generalized as it establishes an approach to studying software



developers' struggles in maneuvering data protection regulations, an ongoing challenge applicable to software teams worldwide.

### Case Descriptions

Both teams apply the four core concepts of Agile: Incrementally designing the software through iterations, instituting ceremonies for inspecting and adapting the product and development process, responding to change collectively, and continuously involving their users [31]. They are structured as typical agile teams with a Team Lead, Product Owner, and team members like developers, testers, and UX designers. Team members were both juniors and seniors, holding between 1–25 years of experience.

Team Travel (12 members) develops and runs an app where travelers can search travel routes and buy tickets across transportation modes like buses, trams, trains, subways, ferries, scooters, and city bikes. They are part of Entur, a public company that provides a digital infrastructure to the Norwegian public transport system. For example, components like payment solutions that travel companies can include in their services, thus relieving them from developing a payment solution themselves. Entur has more than 100 developers organized into 20 development teams, and each team is responsible for a specific part of the digital infrastructure. The teams are described as autonomous, meaning they choose freely how they solve their tasks and rely on development methods. Teams include front-end and back-end developers, designers, and product owners. Entur is inspired by the thoughts behind data mesh [21] and is building a data platform where development teams can publish the data they produce. However, the teams are themselves seen as owners of the data.

Team Welfare develops digital applications used by citizens in need of applying for benefits related to special conditions in life. The new web-based application service is replacing a partly paper-based system. The national welfare administration – NAV – is Norway's largest governmental agency, responsible for distributing 1/3 of the federal budget. NAV has 2000 employees, almost 400 product developers, and 150 product teams organized in 10 product areas. NAV has been at the forefront of working towards becoming data-driven and experimenting with the potential of data utilization. By employing over half of the 800 developers, they went from being dependent on large consultancy companies to being in charge themselves. Insourcing was an essential strategic measure in moving towards a DevOps mindset and going from a few releases a year to more than 1500 a week. NAV is moving towards a decentralized data management model [5] heavily inspired by data mesh [21] which imposes ownership for data that applications produce on the product teams.

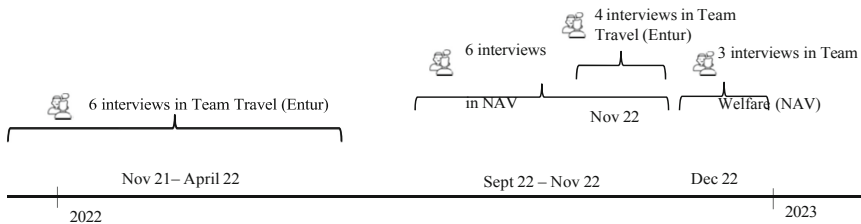
### Research Strategy

We collected data in two rounds of interviews (see Fig. 1). We encountered the privacy issue during the first round of interviews in Team Travel. These interviews were part of a study of development teams' coordination strategies and were reported in a previous paper [32]. Then we explored if privacy issues were also relevant for NAV by interviewing members from different teams and eventually found Team Welfare. The first round of interviews revealed the practical problems of data privacy and gave us an understanding of the context. In the second round, we studied both Team Welfare and Travel's practices for managing data and privacy issues. Interviews were recorded, transcribed, and

coded open-ended using the tool NVivo to identify and structure common codes into larger constructs [33]. Coding and memoing were conducted according to the Constant Comparison Method [34].

To answer the research questions and gain a deep understanding of how data privacy regulations affected the teams' day-to-day operations, we used interviews to pursue detailed stories and descriptions from our respondents. Interviewing allowed us to probe into *why* privacy regulations changed their way of working and *how*. As this topic has increasingly become more and more relevant because the teams started collecting new forms of user data in the last couple of years, it is still a novel and emerging topic for most team members. Interviewing allowed us a window into their ongoing reflections, and it sparked new thoughts in the respondents. This enabled us to compare different perspectives and reflections.

We also analyzed applicable documents like the team's strategy documents and Norway's Digitalization Strategy [35] to explore the research questions thoroughly. For instance, Team Welfare's strategy documents described their team purpose, goals, and development philosophy.



**Fig. 1.** Timeline for data collection of this study

## 4 Findings

We found that the two teams encountered the challenge of assessing if data is sensitive in three situations, namely when collecting, sharing, and storing data. These challenges, however, played out differently due to one central aspect: In seeing their developers' continuous challenges in maneuvering juridical regulations, Team Welfare had met this obstacle by employing a legal advisor assisting both Team Welfare and their "sister team" that worked on another part of the application solution. The two different teams were separate product teams, but still in the same NAV unit, only divided in terms of the different product responsibilities. The legal advisor was particularly engaged in issues dealing with privacy laws and data privacy consequences in their communication with other teams in NAV. Typically, the dialogues took place on Slack, where the different teams exchanged questions, thoughts, and experiences.

However, Team Travel did not have a legal advisor available. They had to evaluate the challenges of data handling independently or seek juridical assistance elsewhere. The following maps out the various challenges public sector product teams encounter regarding data rules and regulations, and the contrasting experiences the two teams reported in this regard.

## 4.1 Collecting Data

As outlined in Sect. 2.1, collecting various data about the public is a central part of the Norwegian welfare system, and in the quest for a data-driven public sector part of this responsibility now falls on the software teams developing the new solutions.

A central concern in that regard had become an often-occurring obstacle for Team Travel, who struggled to assess what data they were allowed to gather, resulting in a “better safe than sorry” mentality. *“We just want to do it right, which limits us [in what data we collect],”* one developer explained. For example, they did not know if they could gather device IDs from user logs and avoided this by gathering user numbers instead. However, only those users who make a purchase or log into their account hold a user number, meaning the team missed out on data from many users. This case describes how collecting data felt risky, and the team accepted lower data quality to lower the risk. There were also other issues; the effort of interpreting privacy regulations was seen as too high risk, and they instead decided to leave the data uncollected.

The wide variety of possible privacy issues developers must consider when gathering data can be described through a question from one of Team Travel’s users. *“A user asked if it was possible for us to identify someone who frequently searched for travel routes from an address to a hospital. In a way, this now becomes sensitive data.”* This was beyond what the team imagined they would have to consider and revealed the additional workload data impedes the development work in form of assessing privacy regulations.

## 4.2 Sharing Data

A central incentive for organizations to become data-driven is the enormous potential and value-creation in data sharing. The Norwegian digitalization strategy 2019–2025 words it as “[w]e must share and reuse more public data, and we must ensure that regulations are digitalization friendly” [35]. For public administrations, data-sharing initiatives are in theory particularly seen as vital for creating common welfare solutions that function across administrative sectors. In practice, however, sharing data also entails major obstacles for software product teams responsible for creating such data-sharing solutions.

The product owner of Team Welfare explained that *what* they can share and *in what way* they share it was an ongoing conversation in the team, especially when they wanted to share historical data with the managers at the various local welfare centers with the incentive of helping them foresee future requests: *“That’s when a lot of legal stuff came up. If they can dig into this, then they can get down to the individual level. In small cases, you can actually locate which user it is, because there may not be that many with the specific needs within a small geographical area”*. This issue was echoed by a Team Welfare developer who referred to what they used as a rule of thumb, namely how data sharing does not abide by data privacy rules if fewer than five people are identified in the material. This “entanglement problem” was especially significant in smaller municipalities where few people applied for the same kinds of aid.

Simultaneously, the teams did not want to own more data than necessary because they did not want to be responsible for it. Gathering data implies a risk of leakage; the more data a team “owns,” the greater the risk. They rather relied on tapping into other

teams' data than gathering it themselves. *"There is a greater risk of leaking sensitive data if it is stored in many places and spread out,"* one developer in Team Travel explained. Tapping into other teams' data implies a willingness to share. Teams then must assess what data they are allowed to share and with whom. *"We talk to other teams to get the data from them [instead of gathering it ourselves] to avoid spreading [ownership of] the data around,"* the developer continued. The uncertainty resulted in hesitation, which led to time delays. It may take months from issuing a sharing request until the data is shared.

A developer in Team Welfare described how owning data was adding extra burden. They often received data requests and had to assess if they could share the data. Requests typically came from other teams that needed business data to develop their own software and managers that used business data to steer resources. *"But we somehow have to assess this all the time, are we allowed to share [the data] with them?"* a developer asked, illustrating their struggles. This was a new problem for the product team. When data was centralized, someone else made this decision; typically, the central data warehouse team. Now, every product team must interpret privacy rules to determine if sharing complies with privacy laws. *"Who is responsible if you share data with someone and they share it further? Is it the people who originally produced the data or is it the consumer?"* the developer continued. To find answers, they must interpret privacy regulations, which made teams anxious. *"I am terrified of a newspaper headline on an individual's sensitive data being leaked. That would be a disaster,"* the developer said. Hesitation seems to be the result, delaying any team that requests data.

Guidelines and instructions are made to support the teams in assessments. However, guidelines are made generic to comply with various domains and contexts. Thus, interpretation is still necessary, which maintains the hesitance of developers. Luckily for Team Welfare, they had a legal advisor supporting them in making these decisions, shortening the time from request to sharing. In contrast, Team Travel was stuck in a world of uncertainty for every incoming request resulting in time-demanding discussions. These findings correspond with NAV's Current Situation Analysis report from October 2021, where one of the central challenges identified moving forward into becoming data-driven was connected to the sharing of data: *"Regulations prevent data sharing and collaboration. The possibility of increased data access and automation is limited by GDPR regulations, other legislations, and system barriers within NAV, between agencies and other actors. This also makes the data quality too poor."*

### 4.3 Storing Data

The Norwegian digital strategy for the public sector states that "[o]pen public data shall be made available for reuse for developing new services and value creation in the business sector." This digitalization incentive is yet another aspect of the datafication process that directly creates increased responsibility for public sector software teams.

For instance, Team Travel struggled to conclude if using Google Analytics complied with privacy regulations. Google Analytics supports teams in gathering and visualizing user metrics (i.e., how many clicks on a button) but requires the data to be uploaded to Google's cloud service. Privacy regulations require specific types of data to be stored within Europe, and Team Travel was in doubt if Google stored data on US servers too.

*“Even though the new Google Analytics 4 is supposed to be much more compliant and such, we don’t really trust it,”* a developer said. They sought answers from the Norwegian Data Protection Authority who told them that they do not consider individual cases, leaving Team Travel in limbo. *“Until we find an alternative tool, we don’t utilize the user metrics.”*

Searching for alternative tools is easier said than done. Data that initially seem harmless can be deceitful when combined with a cloud service. A developer in Team Travel explained that Google Analytics collects cookies on user data which they combine with cookies from other services and build a profile on the user to target advertisement. This means that seemingly harmless and anonymous user metrics turn out to be a potential privacy issue. Team Travel had been searching for alternative tools for more than nine months, with no apparent solution in sight.

In contrast, Team Welfare felt safe in using their Big Query-based (Google Cloud service) in-house service for visualizing user metrics. They did not have to consider if the service was compliant because legal experts had already approved it. Team Welfare still had to consider privacy regulations when uploading data, like Team Travel, but their legal advisor supported them. The big difference between Team Welfare and Travel was that the legal advisor in Team Welfare sat close to the domain, meaning they knew what data user metrics entail in software engineering. Thus, they were willing to give guidelines on individual cases and take on shared responsibility for uploading the data.

In addition, the laws and regulations for storing data also entail that software teams must meet requirements for continuously storing and documenting their *data usage*, and formally document decision processes behind their practices. However, evaluating whether the data can be labeled as sensitive is an ongoing challenge. Oftentimes, if data is considered harmless and only needed for small decisions and not big deliveries, such evaluations will not be formally archived, but rather “hidden” in informal conversations on Slack or Teams. Underlining the difficulties in evaluating the sensitive or non-sensitive nature of data in the context of storing work logs, the Team Welfare legal advisor said: *“I’m not always quite sure when we must document what. [...] So it’s a challenge coming in from outside, from another place in the world, in a way, and into agile development.”*

#### **4.4 Legal Advisor: A Highway to Autonomy for Data-Intensive Teams?**

Although both Team Welfare and Travel encountered various difficulties navigating the different data privacy regulations in their day-to-day work, it was clear that Team Travel experienced more deep-rooted issues, such as not utilizing user metrics due to insufficient legal support. However, inserting a legal advisor into a product team may also come with its own set of challenges. We identified two somewhat intertwined causes of friction in that regard: namely, team synchronizing and interdisciplinary understandings.

Regarding team synchronizing, one central issue was that all the interviewed members on Team Welfare used various communication channels to stay in contact with other team members, other NAV teams and different user groups. Among the most utilized platforms were Teams, Slack and NAV’s internal communication channel for reporting issues, and they had established various group chats with different work topics on the different platforms. Not only did this make it hard to keep track of whom they had talked to and on which channel about which issues; it also made it difficult to track whether

they had documented their data processing according to the rules for how this should be done correctly.

The use of different communication platforms seemed connected to the preferences of people from different disciplinary backgrounds. For instance, Slack was a central communication channel for the technical parts of the team [see 36], whereas the communication with the different case workers took place in closed groups on Teams. Although this point refers to the inclusion of team members from different backgrounds more generally, it was particularly noticeable for Team Welfare, likely due to their team composition being more wide-ranging in terms of interdisciplinary backgrounds.

Additionally, several Team Welfare members pointed out the importance of how synchronizing the team was a longitudinal process, particularly when the team was of an interdisciplinary nature. In the context of the legal advisor, this meant that this person would need to spend a significant amount of time with the team to observe and understand their challenges and everyday practices to truly grasp where assistance would be necessary for the team to operate as efficiently as possible. We will discuss this finding closer in the subsequent section.

In agile software engineering literature, it is well established that there are various barriers to be overcome in interdisciplinary teams for them to become autonomous. That mutual adjustments were challenging was especially articulated by the Team Welfare legal advisor, who said: *“I think [adjustment] is a very big challenge in working interdisciplinary, as we do in the interdisciplinary teams. It’s because we come from different environments. We bring with us very different experiences and knowledge, but we also use words very differently. [...] Then it turns out that we mean very different things when we use the same term”*. The legal advisor mentioned “authorization” and “consent” as often-occurring words, two terms that are central in many contexts related to data handling and storing. They may however have differing connotations for different disciplines.

Although Team Welfare had integrated a legal advisor as part of their team and this person often sat next to the developers when trying to iron out juridical hurdles, the legal advisor seemed to have little understanding of how much the data regulations impacted the developers’ day-to-day tasks. It was clear that the legal advisor seemed to believe the developers solely care about effective technical solutions, not societal responsibility:

*There is a very basic thought I get from time to time, that we bring in developers who think it is exciting to work with an idea that ‘we can just make a shopping cart’ where they can sort of get to pick the things they want. And then, the reality is far from that [...] It is, after all, an administrative responsibility that lies at the bottom of everything we do, which must be communicated from scratch.*

The legal advisor thus doubted the software developers’ ability to grasp the larger picture of their work, which seemed far from the developers’ own stories in that regard. As a Team Welfare developer put it: *“[Juridical issues] is something that everyone should be aware of, it is not like you should always think that this is something that the lawyers on the team handle and are responsible for. Everyone must consciously evaluate what kind of data we store, whether we have the right to handle that data, whether we really need to check the legal basis more carefully before we do anything.”* However, it was also

evident that with proper team synchronizing came mutual adjustments and increased interdisciplinary understandings throughout the entire product team-user chain. And despite the various challenges with data regulations and usage, the Team Welfare product owner underlined the importance of not being blindsided by the juridical hurdles, always keeping in mind how data actually promotes the development of software products: *“I really think that the most important aspect with data for us is knowing that it is the right product that we are making, and somehow finding confirmation of that and seeing that it provides the value that our project is meant to have”*. The same team’s legal advisor concluded: *“For me, I think the keyword is interdisciplinary understanding. At least, that is a keyword for developing good solutions.”*

## 5 Discussion and Implications for Practice

With public sector organizations globally being in the midst of data-driven “transformation journeys” [37], inevitably, both the software developer role and the composition of product teams are changing rapidly. We here return to our research questions: How are data privacy regulations affecting the day-to-day work of product teams? And what are the pros and cons of including a legal advisor as part of the team to overcome juridical hurdles? In answering these, we discuss the direct implications datafication has on interdisciplinary agile product teams’ practices, focusing first on how laws regulating data collection, sharing, and storing data expand and complexify software developers’ responsibilities.

Data work has not traditionally been a part of the software developer role. Amidst the quest towards becoming data-driven, organizations – both public and private – have added new areas of competence to the developer profession – as well as new liabilities: Carefully considering ethical and juridical aspects of using data now permeate software developers’ day-to-day tasks, and for both teams being responsible for data had become a burden as much as a realm of possibility. Although decentralizing data storage, in theory, gives product teams more control and the ability to self-organize, the context-dependent and ambiguous laws regulating data collection, storage and sharing meant that the potential value-creation of data brought about ambivalent feelings for the teams.

To realize the potential benefits of datafication in the public sector, data needs to directly reflect the public. However, knowing that any minor misuse of personal data could lead to newspaper headlines, developers are forced to settle for fewer data and data of lesser quality than optimal to avoid breaking potential privacy regulations. Consequently, the teams are forced to make decisions and deliver continuous updates based on a skewed picture of their users, i.e., the Norwegian public. Ultimately, the organizations risk investing huge resources in becoming data-driven only to make the day-to-day tasks of developers more demanding and create a false impression of being driven by “real world” data. This corresponds to and adds further layers to the findings of Broomfield and Reutter [14] whose “search for the citizen” in data-driven public administrations find that civil society tends to be obscured in the data and reduced to passive users in datafied public administrations.

Another vital finding in this respect is that the classification of data as either “private” or “sensitive” is not straightforward in a data-intensive software development context.

As the teams pointed out, when faced with large amounts of data that were seemingly risk-free, they oftentimes found themselves in a situation where the *combination* of these data in fact created potential data privacy violation issues. As the study of data democratization is in its early phase [6], more studies are needed here both in order to find better ways of classifying data and in terms of empirical studies on software teams' experiences with data classification.

Zooming in on such situations in the teams we studied, our findings suggest that the team working closely with a legal advisor was better equipped for responding to data privacy regulations. And although there were also challenges involved with inserting a team member whose responsibility was the juridical side of data-driven software development, Team Welfare had slowly but surely developed tactics to make the team more autonomous. For instance, the legal advisor underlined the importance of being "close by" the technical sides of the teams to "grasp" the issues when they emerged, an understanding that was shared by the whole team.

Interestingly, the Team Welfare legal advisor saw it as a large responsibility to educate the technologists about the rules and regulations they must operate within. This responsibility, however, was also well-engrained into the minds of the developers. So, despite the interdisciplinary nature of the team, the understanding of others' ethical responsibility regarding data handling was not communicated that well between the group members from different disciplines, perhaps due to the different terms used by the various disciplines.

Another important point we draw from the findings in this regard is that it is not enough for the legal advisor to join the team for short sessions now and then: It takes time to synchronize the interdisciplinary teams to be as autonomous as possible and for the legal advisor to develop a contextual understanding of the data product teams and their day-to-day practices and challenges. Several informants pointed out how the teams must "mature together" over a longer period of time.

Our study shows that there likely are many benefits of including a legal advisor in data-intensive product teams, as it undoubtedly makes them more confident in their day-to-day data handling practices. However, as previous studies also have pointed to [23], expanding software development teams and including new, formal roles beyond the technical team requires much effort in synchronizing and overcoming interdisciplinary barriers. The additional financial costs of including legal experts must also be examined. We find, however, that the costs of *not* having continuous access to a legal advisor can likely be much larger, considering that the outcome may be week-long – perhaps even month-long – delays in deliveries, as well as mishandling user data. This especially holds true for organizations managing large amounts of data that are difficult to evaluate as sensitive or not, and the stakes are high in terms of potential misuse of the data in question. We thus especially recommend such teams to consider the possibility of including a legal expert as part of the team.

## 6 Conclusion and Recommendations for Future Work

The increasing regulations and complexities around datafication processes have expanded the responsibilities and expectations of software developers. To understand how agile teams deal with privacy regulations when becoming data-driven, we investigated two agile product teams. One team had a legal advisor as one of the team members,



and one team did not, keeping this expertise outside the team. Both teams struggled with knowing which data they were allowed to collect and share. We found that having a legal advisor on the team shortened the time to solve privacy issues. The team without this role had time-demanding discussions within the team, and experienced uncertainty and hesitation to act, which resulted in privacy issues taking months to solve. While it was beneficial to have a legal advisor, challenges included coordination and syncing issues among the team members. For example, because they were from different domains, they had different preferences for communication tools (the technical team members preferred to communicate on Slack). Coming from different backgrounds also meant that they had different understandings of terms (e.g., the term “authorization”), which potentially could lead to misunderstandings in the team.

With both public and private sector software projects focusing increasingly on the possibilities of data-driven development, it is vital that also the software engineering research field reflects this impending paradigm shift. Future work should hence monitor closely how datafication affects software developers and their increasingly interdisciplinary teams to pinpoint the implications these changes have on day-to-day practices. Further studies are especially needed addressing how the principle of continuous compliance can create significant hurdles for software development and how this best can be overcome for teams also in various parts of the private sector.

As this study addresses only two teams within the Norwegian public sector, it is hence limited in the sense that it cannot be generalized to other contexts. We thus recommend future work to address these issues with a quantitative or mixed-methods approach, that can show a broader scope of how software developers and product teams tackle data protection regulations. Additionally, as digitalization processes also take place outside Western societies, however receiving considerably less academic attention, future studies addressing how software developers across the democratic world are seeking ways to tackle these issues would be especially welcoming for broadening the scholarship.

**Acknowledgments.** We wish to thank NAV, Entur and the informants for willingly sharing their experiences. Also, we thank Knowit AS and the Norwegian Research Council for funding the research (grant number 321477).

## References

1. van Ooijen, C., Ubaldi, B., Welby, B.: A data-driven public sector: enabling the strategic use of data for productive, inclusive and trustworthy governance. OECD, Paris, May 2019. <https://doi.org/10.1787/09ab162c-en>.
2. Andreassen, R., Kaun, A., Nikunen, K.: Fostering the data welfare state: a Nordic perspective on datafication. *Nord. Rev.* **42**(2), 207–223 (2021). <https://doi.org/10.2478/nor-2021-0051>
3. Dencik, L., Kaun, A.: Datafication and the welfare state. *Glob. Perspect.* **1**(1) (2020). <https://doi.org/10.1525/gp.2020.12912>
4. Reutter, L.: Constraining context: situating datafication in public administration. *New Media Soc.* **24**(4), 903–921 (2022). <https://doi.org/10.1177/14614448221079029>

5. Vestues, K., Hanssen, G.K., Mikalsen, M., Buan, T.A., Conboy, K.: Agile data management in NAV: a case study. In: Stray, V., Stol, K.J., Paasivaara, M., Kruchten, P. (eds.) *Agile Processes in Software Engineering and Extreme Programming*, pp. 220–235. Springer, Cham (2022). [https://doi.org/10.1007/978-3-031-08169-9\\_14](https://doi.org/10.1007/978-3-031-08169-9_14)
6. Kashyap, B., Shinghal, K.: Data democratisation: the evolution of informed decisions (2019). <https://doi.org/10.13140/RG.2.2.33103.51361>
7. Stray, V., Moe, N.B., Hoda, R.: Autonomous agile teams: challenges and future directions for research. In: *Proceedings of the 19th International Conference on Agile Software Development: Companion*, Porto Portugal, pp. 1–5, May 2018. <https://doi.org/10.1145/3234152.3234182>
8. Hirsch, D.D.: The glass house effect: big data, the new oil, and the power of analogy. *Big Data* **66**
9. Klotins, E., Peretz-Andersson, E.: The unified perspective of digital transformation and continuous software engineering. In: *2022 IEEE/ACM International Workshop on Software-Intensive Business (IWSiB)*, pp. 75–82, May 2022. <https://doi.org/10.1145/3524614.3528626>
10. Jahns, V.: Data fabric and datafication. *ACM SIGSOFT Softw. Eng. Notes* **47**(4), 30–31 (2022). <https://doi.org/10.1145/3561846.3561854>
11. Fernández-Rovira, C., Valdés, J.Á., Molleví, G., Nicolas-Sans, R.: The digital transformation of business. Towards the datafication of the relationship with customers. *Technol. Forecast. Soc. Change* **162**, 120339 (2021). <https://doi.org/10.1016/j.techfore.2020.120339>
12. van Dijck, J.: Datafication, dataism and dataveillance: big data between scientific paradigm and ideology. *Surveill. Soc.* **12**(2), 197–208 (2014). <https://doi.org/10.24908/ss.v12i2.4776>
13. What is Datafication? Definition from Techopedia.com. <http://www.techopedia.com/definition/30203/datafication>. Accessed 03 Feb 2023
14. Broomfield, H., Reutter, L.: In search of the citizen in the datafication of public administration. *Big Data Soc.* **9**(1), 20539517221089304 (2022). <https://doi.org/10.1177/20539517221089302>
15. Misuraca, G., Van, N.C.: AI Watch - Artificial Intelligence in public services. JRC Publications Repository (2020). <https://publications.jrc.ec.europa.eu/repository/handle/JRC120399>. Accessed 03 Feb 2023
16. Ruppert, E., Isin, E., Bigo, D.: Data politics. *Big Data Soc.* **4**(2), 2053951717717749 (2017). <https://doi.org/10.1177/2053951717717749>
17. Tupasela, A., Snell, K., Tarkkala, H.: The Nordic data imaginary. *Big Data Soc.* **7**(1), 2053951720907107 (2020). <https://doi.org/10.1177/2053951720907107>
18. Sadowski, J.: When data is capital: datafication, accumulation, and extraction. *Big Data Soc.* **6**(1), 2053951718820549 (2019). <https://doi.org/10.1177/2053951718820549>
19. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*: Kleppmann, Martin: 9781449373320: Amazon.com: Books. <https://www.amazon.com/Designing-Data-Intensive-Applications-Reliable-Maintainable/dp/1449373321>. Accessed 03 Feb 2023
20. Dehghani, Z.: *Data Mesh: Delivering Data-Driven Value at Scale*, 1st edn. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly Media (2022)
21. *Data Mesh Principles and Logical Architecture*. martinowler.com. <https://martinowler.com/articles/data-mesh-principles.html>. Accessed 03 Feb 2023
22. Frontpage. Datatilsynet. <https://www.datatilsynet.no/en/>. Accessed 03 Feb 2023
23. Moe, N.B., Stray, V., Hoda, R.: Trends and updated research agenda for autonomous agile teams: a summary of the second international workshop at XP2019. In: Hoda, R. (ed.) *Agile Processes in Software Engineering and Extreme Programming*, pp. 13–19. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-30126-2\\_2](https://doi.org/10.1007/978-3-030-30126-2_2)

24. Hukkelberg, I., Berntzen, M.: Exploring the challenges of integrating data science roles in agile autonomous teams. In: Hoda, R. (ed.) XP 2019. LNBP, vol. 364, pp. 37–45. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-30126-2\\_5](https://doi.org/10.1007/978-3-030-30126-2_5)
25. Klotins, E., Gorschek, T., Sundelin, K., Falk, E.: Towards cost-benefit evaluation for continuous software engineering activities. *Empir. Softw. Eng.* **27**(6), 157 (2022). <https://doi.org/10.1007/s10664-022-10191-w>
26. Alhazmi, A., Arachchilage, N.A.G.: I'm all ears! Listening to software developers on putting GDPR principles into software development practice. *Pers. Ubiquit. Comput.* **25**(5), 879–892 (2021). <https://doi.org/10.1007/s00779-021-01544-1>
27. Stray, V., Fægri, T.E., Moe, N.B.: Exploring norms in agile software teams. In: Abrahamsson, P., Jedlitschka, A., Nguyen Duc, A., Felderer, M., Amasaki, S., Mikkonen, T. (eds.) *Product-Focused Software Process Improvement*, pp. 458–467. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-49094-6\\_31](https://doi.org/10.1007/978-3-319-49094-6_31)
28. Yin, R.K.: *Applications of Case Study Research*. SAGE (2011)
29. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. *Empir. Softw. Eng.* **14**(2), 131 (2008). <https://doi.org/10.1007/s10664-008-9102-8>
30. Oates, B.J.: *Researching information systems and computing*. Sage (2005)
31. Baham, C., Hirschheim, R.: Issues, challenges, and a proposed theoretical core of agile software development research. *Inf. Syst. J.* **32**(1), 103–129 (2022)
32. Sporse, T., Moe, N.B.: Coordination Strategies When Working from Anywhere: A Case Study of Two Agile Teams. arXiv, 08 April 2022. <https://doi.org/10.48550/arXiv.2204.03978>
33. Technische Universität München et al.: Grounded theory methodology in information systems research. *MIS Q.* **41**(3), 685–701 (2017). <https://doi.org/10.25300/MISQ/2017/41.3.02>
34. Seaman, C.B.: Qualitative methods in empirical studies of software engineering. *IEEE Trans. Softw. Eng.* **25**(4), 557–572 (1999). <https://doi.org/10.1109/32.799955>
35. K. moderniseringsdepartementet: Én digital offentlig sektor: Digitaliseringsstrategi for offentlig sektor 2019–2025. Regjeringen.no, 14 June 2019. <https://www.regjeringen.no/no/tema/statlig-forvaltning/ikt-politikk/digitaliseringsstrategi-for-offentlig-sektor/id2612415/>. Accessed 06 Feb 2023
36. Stray, V., Moe, N.B., Noroozi, M.: Slack me if you can! Using enterprise social networking tools in virtual agile teams. In: 2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE), pp. 111–121 May 2019. <https://doi.org/10.1109/ICGSE.2019.00031>
37. Stettina, C.J., van Els, V., Croonenberg, J., Visser, J.: The impact of agile transformations on organizational performance: a survey of teams, programs and portfolios. In: Gregory, P., Lassenius, C., Wang, X., Kruchten, P. (eds.) XP 2021. LNBP, vol. 419, pp. 86–102. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-78098-2\\_6](https://doi.org/10.1007/978-3-030-78098-2_6)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



# **Short Paper**



# Real-Life Water-Scrum-Fall: Insights from Large Companies in Czech Republic

Michaela Křivánková  and Daniel Remta <sup>(✉)</sup> 

Prague University of Economics and Business, Prague, Czech Republic  
xremd03@vse.cz

**Abstract. Background.** A combination of traditional and agile methodologies into so-called “hybrid approaches” are today’s reality. The intention is to attempt maximization of benefits from both approaches. Water-Scrum-Fall is a methodology for hybrid approaches. However, there are no specific guidelines described for Water-Scrum-Fall. **Aim.** Explore and describe the reality of Water-Scrum-Fall in selected large companies in the Czech market. **Method.** Exploratory multiple-case study – four companies, eight participants. Semi-structured interviews and thematic analysis. **Results.** Insights into real-life Water-Scrum-Fall including involved roles, associated events, and typically created artifacts. **Conclusion.** In examined companies, modified agile methodologies are used as part of Water-Scrum-Fall, while their benefits are not fully unlocked due to poor implementations. However, project success rate is high. Hybrid approaches still require rigorous analysis before the development is started, which strongly limits the agility in the following development phase. Customer involvement or feedback loop is omitted in the development process. We accompany poor implementations with the unpreparedness of companies to fully adopt Agile, and missing senior roles, like scrum masters, to educate teams and organizations in proper agile practices.

**Keywords:** Water-Scrum-Fall · Hybrid Methodologies · Agile

## 1 Introduction

In almost every project, some type of methodology or combination of methodologies is followed, although we may not be aware of it [1]. A methodology is a structured approach to solving a problem [2].

In software development, two types of methodologies are mainly used – traditional and Agile [3]. Traditional (or rigorous) methodologies are often voluminous and are based on the assumption that the processes involved in building an IS/ICT can be described, planned, managed and measured – one of the most widely used traditional methodologies is Waterfall [4], where one phase needs to be finished before moving to another phase. Agile methodologies, where several phases run at the same time, are based on unpredictability and responsiveness to rapidly changing requirements. Therefore, the main difference between these methodologies is in the ability to adapt to changes [4].

Many companies are using the so called “hybrid approach” to maximize benefits from both methodologies [5]. Kuhrmann et al. [6] defined the hybrid approach as any

combination of Agile and traditional approaches that an organizational unit adopts and adapts to its own needs.

One of these hybrid approaches is the Water-Scrum-Fall described by West [7]. There are no specific guidelines for Water-Scrum-Fall, so the implementation of the methodology itself can vary from company to company. Furthermore, organizations often end with following Water-Scrum-Fall unintentionally without recognizing it [7].

The aim of our research was to describe the reality of Water-Scrum-Fall. We focused on large corporate companies (with more than 5000 employees) within the Czech market that are more inclined to use this approach [6]. In this paper, the following research questions are addressed: 1) What does Water-Scrum-Fall phases consist of in selected companies in the Czech market? 2) Which roles are part of the Water-Scrum-Fall phases in selected companies in the Czech market?

Our research helps to shape the definition of Water-Scrum-Fall and brings real-life examples. Additionally, our research can help organizations with adoption of the hybrid-approaches that improve the agility of organizations [9].

## 2 Related Work

In this section we present related literature that represent the current state of art. The intention is to introduce the reader into the context and the current state of art.

Kuhrmann et al. [6] focused on development approaches used in practice and triggers for hybrid approaches. Kuhrmann et al. [6] described that large and very large companies as well as start-ups tend to use hybrid methodologies for better risk management. The reasoning behind the change to hybrid methodologies can have various drivers [8]. Hartman et al. [9] described that companies are using hybrid methodologies as a step towards agile organization.

Studies [6, 8, 10] are discussing how companies that switched to using hybrid methodology reached similar results. The methodology was developed, adapted and improved over time and based on gained experience. Only around 20% of companies switched to hybrid methodologies based on a plan and newly created processes [6, 8].

Water-Scrum-Fall is a term that was coined by West [7]. West described the three phases of Water-Scrum-Fall. The first (**Water**) and third phase (**Fall**) are based on the Waterfall model [11]. In **Water** - upstream project planning takes place, and plans for time, budget, and scope management are set up. Additionally, user requirements and system requirements are created. Next, the development phase (**Scrum**) is based on Scrum [12]. In **Scrum** phase – design, development and implementation is done in iterative steps by the development team. **Fall** - when the development team has implemented all requirements, the solution is delivered using the traditional procedure based on establishing quality control gates to reduce the frequency of software releases [7].

Reiff and Schlegel [13] described Water-Scrum-Fall as a good introduction for companies that have been using the traditional approach and are now moving towards Agile. They concluded that organizations with well-structured processes with systematic milestones are suitable for the implementation of a hybrid approach. The maximization of project success was mentioned as one of the main advantages.

Butgereit [14] discussed five lessons learned during the implementation of Water-Scrum-Fall for a project in South Africa. These lessons were incorporated in the interview guide for our research.

In the research conducted by PMI [15], about 20% of respondents stated that they are using hybrid methodologies. Menčík et al. [16] focused on the state of Agile in the Czech Republic, in which 16 companies out of 209 marked the combination of Waterfall and Scrum as their used methodology. However, the primary focus of the study [16] was on Agile adoption.

Overall, there is little known about the real-life implementations of Water-Scrum-Fall, and calls for additional empirical research were made [8, 13].

### 3 Research Method

This section describes study context, data collection process, and analysis. Our study was based on an exploratory multiple-case study [17]. Eight interviews in four companies were conducted, and the results were analyzed using thematic analysis.

**Context.** For the selection of participants, purposive snowballing sampling was used [18]. Snowball sampling involves asking participants for recommendations of acquaintances who might qualify for participation, leading to “referral chains”. The snowballing gave us access to four companies and 8 participants. Each company has over 5000 employees and belongs to highly-regulated sectors.

All companies were using Waterfall before including Agile practices, which moved them to a currently prevailing hybrid approach. Company A has been using this approach for approximately 7 years, company B for 4 years, and companies C and D for over 3 years.

During the study, in companies A, B, and D participants were involved with small projects and sub-projects of a bigger product. In company C participants worked on continuous product development. A, B, and D had the teams fluid, always rebuild around specific project. The team from company C was stable. We were not able to get many insights to the current projects the participants were involved with, but for A and B process automation was mentioned. The team from C was working on implementation of the changes to better comply with GDPR. When interviewing participants from the same company, people from different roles or departments and teams were selected. The background of each participant is presented in Table 1.

**Data Collection.** Interview Protocol was created per [19] and used to guide the semi-structured interview. For the interviews, each participant was first asked about their background and context. Then, seven main themes were defined, each consisting of questions adhering to their sub-themes.

An overview of the main themes and their sub-themes is given below in Table 2.

Interviews were done through the video conferencing tools Google Meet or Microsoft Teams, recorded, and transcribed verbatim. The interview lengths ranged from 23 to 50 min.



**Table 1.** Participants in the study

Participant ID	Company	Sector	Position
P1	A	Biotechnology company	Team leader
P2	A	Biotechnology company	Team leader
P3	A	Biotechnology company	Developer
P4	B	Energy company	Project manager
P5	B	Energy company	Team leader
P6	C	Services for carriers	Project manager
P7	C	Services for carriers	Scrum Master
P8	D	Bank	Project manager

**Table 2.** Themes for interview

Theme name	Sub-themes
Size, duration	Size of teams, Scope of projects (duration)
“Water”	Project planning, Requirements collection
Scrum	Roles, Sprint planning, Sprint length, Daily stand-ups, Sprint review, Retrospective, Documentation
“Fall”	Method of project continuation, Length of deployment, Follow-up meetings with business
Project success rate	Percentage success rate, Role of methodology in success rate
Evaluation	Advantages, Disadvantages
Future	Plans to change the methodology

**Data Analysis.** Thematic analysis [20] was used to analyze the data. The transcripts were subsequently coded in QDA Miner Lite. First, the central themes were chosen, of which there are seven in total. The identified central themes were: Size, Duration; “Water”, Scrum, “Fall”, Project success rate, Evaluation, and Future. Next, for each central theme we identified sub-themes. Sub-themes emerged during the data analysis. The themes and sub-themes are described in Table 3.

**Table 3.** Thematic analysis themes and identified sub-themes

Theme	Identified sub-themes
Size, duration	Size of teams, Scope of projects (duration)
“Water”	Project planning, Requirements collection, Pre-project documentation
Scrum	Project manager as Scrum master, The length of Sprints, Daily-stand-ups, Omitted retrospective, Failure to fulfill Product Owner role, Mis-alignment between business and IT, Roles in development team
“Fall”	Project acceptance, The length and frequency of deployment, Method of project continuation, Follow-up meetings with business
Project success rate	High success rate, User resistance
Evaluation	Disadvantages of Waterfall part, Disadvantages of Scrum part, Advantages of Waterfall part, Advantages of Scrum part, Limitations in agile approach
Future	Transition to SAFe, Transition to agile

## 4 Results

In this section, we present distillate for each investigated theme, supported by evidence gathered during the interviews in the form of quotations. Every quote is referenced to the corresponding participant from Table 1. All the interviewees confirmed that their organizations attempted to become agile. None of the interviewees explicitly said that they are following Water-Scrum-Fall. However, the results showed that Water-Scrum-Fall was the reality.

**Size, Duration.** The duration of projects varied from 3 months to a few years. On average, the projects are about 6 months long. *“Year is probably the longest time, but usually six months is optimal.” [P1]* The teams are rather small and usually consisted of 5 people in order to maintain easy communication. *“The less people, the better. Ideally, we’re talking about maybe a team of five people.” [P5]*

**“Water”.** After project initiation, requirement gatherings usually take up to 3 months. *“It can be anywhere from one to three months.” [P8]* Roles such as customer, analyst and architect are involved. *“So first the business brief is written, then it goes to the analyst and architect.” [P4]* There is always at least one document that is created that contains very detailed specifications of the software to be developed, including flowcharts and server specifications. *“[...] there has to be a document that is called the Business Specification and Solution Blueprint, and it actually describes what exactly needs to be done, in great detail.” [P8]*

**Scrum.** The development follows the Scrum methodology [12], which is being locally modified. Often the project manager is involved, acting as both project manager and Scrum Master. *“[...] so we do not have a Scrum Master. Basically I do it, because I do the meetings, but otherwise I don’t really do anything physical in terms of projects.” [P1]* The customer acts as the Product Owner, a role that is not always followed correctly.

*“The role that we struggle with a little bit is Product Owner, often he is formally defined, but people are still learning how to use it.” [P2] The development team consists of analysts, testers and developers, “[...] we also have Subject Matter Experts, which is somewhere between a tester and an analyst.” [P6]*

The largest modifications can be observed in Scrum events, especially in Daily stand-ups and retrospectives. Daily stand-ups vary in their length, and most importantly, they are not always done on a daily basis. *“We have stand-ups once every two days - Monday, Wednesday, Friday - and it lasts about 30 min.” [P6]* As for retrospectives, if the team does not feel the need to conduct them, or if there is not enough feedback, they are often skipped or dealt with in other events, such as during the daily stand-up. *“We don’t have a retrospective, probably we haven’t even noticed the need to have one yet.” [P3]*

**“Fall”.** Documentation is usually written after the software (or at least the first version) is developed, and is usually created by updating the detailed specification document that was prepared during the Analysis phase. *“[...] of course not everything will be created at the moment when it should be created, but at the latest during the phase Administrative Close it will be finished.” [P2]* There is always some sort of user acceptance before deployment; in some cases, the acceptance is based on acceptance criteria, and in other cases the acceptance is done by the Product Owner. As far as the timing of deployment is concerned, usually the development team itself can’t influence the timing. In all selected companies there are explicit “release windows” when deployment can happen. Release windows are set by the Release Manager. *“The releases are company-wide, so they’re once a month, meaning it’s not after every sprint.” [P7]*

**Project Success Rate.** In selected companies, projects are considered very successful. P4, P2 and P8 stated that almost 100% of their projects are delivered on time and on budget; however, these attributes are moved by change requests, so the latest deadline is often different to the originally-expected one. Additionally, a certain user resistance was encountered. Users try to use the new solution minimally or avoid using the solution completely. *“Users tend to boycott anything new.” [P3]*

**Evaluation.** One of the disadvantages is that a large change can affect the entire project plan. *“When the projects are for a year, it’s very hard to hit a specific month if you have a change that wasn’t planned for.” [P5]* Another disadvantage was connected to the mindset associated with Agile – people involved in projects were not able to be self-organized. *“They have to be proactive. There are high demands on human qualities.” [P1]* Participants articulated the unpreparedness for change in the company as a whole. *“That organization has to be ready for it, and our organization as a whole is not ready for it. [...] we have a methodology for that, but it just doesn’t work 100%.” [P4]* Yet, the teams seemed to benefit from the introduction of the Scrum phase. *“[...] it’s more fun and more lively, more active, if I see some results in 14 days.” [P7]*

**Future.** All companies in the study are using Water-Scrum-Fall, except for company C, which is already switching towards the agile way of working. However, the transition phase was described as “painful” and “chaotic” by both participants P6 and P7. In company A, we observed an approach which could be called Water-SAFe-Fall. There were many projects with approximately 100 people involved. Company A wants to

transform to an agile company using SAFe, but the structure and culture are not prepared for the transformation yet. *“So I think there is still a big pitfall and a big piece of work that especially the big corporations have to do, and that is to prepare the culture, the mentality of that big environment so that agile can work in it.”* [P2] Companies B and D are not planning to change current methodology anytime soon.

#### 4.1 Roles, Events, Artifacts in Water-Scrum-Fall

Based on the conducted interviews, we have enhanced the Water-Scrum-Fall figure presented by Reiff and Schlegel [13] with discovered roles, events and artifacts. The result is visible in Fig. 1. Additionally, we have added the “Post-completion” phase, as all the participants mentioned team participation in increased support after the release. Furthermore, in this phase the teams are gathering feedback for their solutions. *“[...] after they are using the product for some time we come and ask “Are you happy with the solution, do you need anything more?”* [P2] The intention is to identify *“[...] if there is a need to change anything, or support the change anyhow, so that the solution could be used correctly”* [P3]. In case changes are needed, the process has to be restarted from the Water phase.

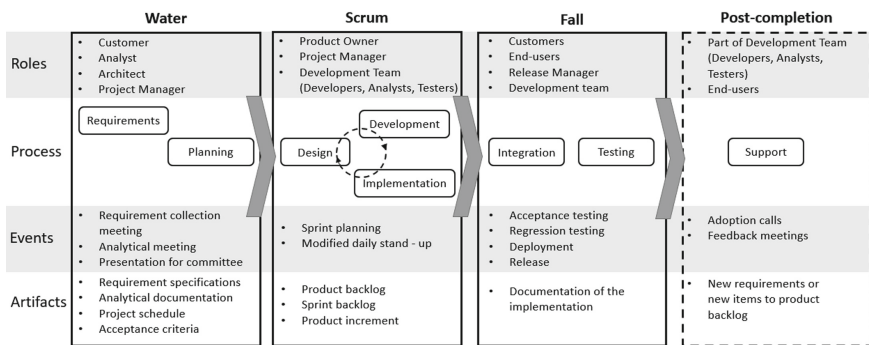


Fig. 1. Reality of Water-Scrum-Fall

## 5 Discussion

Gathering insights into Scrum-Water-Fall from large companies in the Czech Republic, we collected a data set containing qualitative data from four large companies through interviews with eight participants.

The examined companies tend to move towards agile methodologies. Aligned with Hartman et al. [9] findings, companies mix traditional approaches with agile methodologies as the first step towards an agile organization. Generally, the hybrid model was very similar in all companies. We saw the difference in the Scrum phase where tailoring of the Scrum roles and events occurred.

Participants reported struggles with combining traditional detailed up-front planning and detailed analysis [4] with the ability to adapt to changes, which is connected with Agile [4]. “[...] *there has to be a document that is called the Business Specification and Solution Blueprint, and it actually describes what exactly needs to be done, in great detail.*” [P8] The specification “**Water**” phase is followed by the actual development “**Scrum**” phase. Teams are following modified versions of Scrum [12]. The focus on team synchronization (daily stand-ups) and continuous improvement (retrospectives) is minimized or omitted. “*We don’t have a retrospective, probably we haven’t even noticed the need to have one yet.*” [P3] None of the participants mentioned sprint reviews, close collaboration with the customers, or another feedback loop to be happening during the development phase. Some form of acceptance testing is conducted only at the end of development during the “**Fall**” phase, and the actual feedback from end-users is gathered in the Post-completion phase.

The poor Scrum implementations suggest that respective teams haven’t been properly familiarized with or educated about the ideas and benefits of agile methodologies. Still, even the partial implementation seemed to bring positive effects on the team. We accompany poor implementations with missing senior roles, like scrum masters, to educate teams and organizations in proper agile practices. “[...] *so we do not have a Scrum Master. Basically I do it, because I do the meetings, but otherwise I don’t really do anything physical in terms of projects.*” [P1].

West’s [7] assertion that Water-Scrum-Fall is a reality for many organizations today, whether intentionally or unintentionally, was confirmed in our research. Similarly the results confirmed improvement in project success rate, described as one of the benefits of hybrid approaches in [13]. However, participants considered only delivery on time and on budget as the success factors. The evaluation of the solution usefulness is typically happening in the post-completion phase, and for potential changes the Water-Scrum-Fall has to be restarted. This limits the agility in the software development.

All examined organizations come from domains that are subject to regulations and domain standards. These constraints often force them to work in a way that is not purely agile but may require traditional processes. Hence, Water-Scrum-Fall could be the answer. Still, it seems that companies would like to achieve additional benefits reported with agile approaches but are not willing to perform big changes [6]. “*That organization has to be ready for it, and our organization as a whole is not ready for it. [...] we have a methodology for that, but it just doesn’t work 100%.*” [P4] The discovered reality of Water-Scrum-Fall suggests that organizations just bring in some of the prescribed events and roles from Scrum [12], but tailor the methodology to fit their existing approaches, which results in the similar Water-Scrum-Fall as before.

**Conclusion.** Our study revealed that Water-Scrum-Fall was a reality in the examined organizations and described the content of phases and roles in large companies in the Czech Republic. Modified agile methodologies are used in Water-Scrum-Fall. Hybrid approaches still require rigorous analysis before the development is started, which strongly limits the agility in following development. Customer involvement or feedback loop is omitted in the development process. We accompany poor implementations

with the unpreparedness of companies to fully adopt Agile and missing senior roles, like scrum masters, to educate teams and organizations in proper agile practices.

**Limitations.** The general limitation of qualitative research is difficulty in generalizing. Additionally, our sample was influenced by snowball sampling, which basically located a source of potential participants who are convenient in their proximity and willingness to participate. Still, it brings in-depth view of the research area. Our study was conducted only in companies participating in the Czech market, therefore, it may be impacted by local bias. Despite the limited number of participants, we believe that our study provides valuable insights into real-life Water-Scrum-Fall implementations.

**Future Work.** More empirical research is needed to confirm findings from this study in different environments, companies of different sizes, and from various sectors.

**Acknowledgment.** This work was supported by an internal grant funding scheme (F4/35/2022) administered by the Prague University of Economics and Business.

## References

1. Charvat, J.: Project Management Methodologies: Selecting, Implementing and Supporting Methodologies and Processes for Projects. Wiley (2003)
2. OECD: OECD Glossary of Statistical Terms - Methodology Definition (2001). <https://stats.oecd.org/glossary/detail.asp?ID=1652>. Accessed 6 Feb 2023
3. Grove, C.: Project Management & Agile Methodologies. <https://www.cprime.com/resources/blog/project-management-agile-methodologies/>. Accessed 6 Feb 2023
4. Javanmard, M., Alian, M.: Comparison between agile and traditional software development methodologies. *Sci. J. (CSJ)* **36** (2015)
5. Reiff, J., Schlegel, D.: Hybrid project management – a systematic literature review. *Int. J. Inf. Syst. Proj. Manag.* **10**, 45–63 (2022)
6. Kuhrmann, M., et al.: Hybrid Software and system development in practice: waterfall, scrum, and beyond. In: ICSSP (2017)
7. West, D.: Water-Scrum-Fall Is The Reality of Agile for Most Organizations Today. Forrester (2011)
8. Kuhrmann, M., et al.: Hybrid software development approaches in practice: a European perspective. *IEEE Softw.* **36**, 20–31 (2019)
9. Hartman, B., et al.: What is Hybrid Agile, Anyway? <https://www.agilealliance.org/what-is-hybrid-agile-anyway/>. Accessed 6 Feb 2023
10. Linders, B.: Delivering Software with Water-Scrum-Fall. <https://www.infoq.com/articles/delivering-software-water-scrum-fall/>. Accessed 6 Feb 2023
11. Royce, W.W.: Managing the development of large software systems. In: Proceedings, IEEE WESCON (1970)
12. Schwaber, K., Sutherland, J.: Scrum Guide. <https://scrumguides.org/scrum-guide.html>. Accessed 6 Feb 2023
13. Reiff, J., Schlegel, D.: Hybrid project management – a systematic literature review. *Int. J. Inf. Syst. Proj. Manag.* **10**(2), 45–63 (2022)

14. Butgereit, L.: Five lessons learned using water-scrum-fall in South Africa. In: Proceedings of the 2018 International Conference on Multidisciplinary Research (2021)
15. PMI: Pulse of the Profession 2017. <https://www.pmi.org/learning/thought-leadership/pulse/pulse-of-the-profession-2017>. Accessed 6 Feb 2023
16. Menčík, M., Buchalceková, A., Doležel, M., Koudelka, J.: Kvantitativní průzkum stavu využívání agilních přístupů pro dodávku softwaru v České republice. (2019)
17. Yin, R.K.: Case Study Research: Design and Methods. SAGE (2009)
18. Robinson, O.C.: Sampling in interview-based qualitative research: a theoretical and practical guide. *Qual. Res. Psychol.* **11**(1), 25–41 (2014)
19. Kallio, H., Pietilä, A.-M., Johnson, M., Kangasniemi, M.: Systematic methodological review: developing a framework for a qualitative semi-structured interview guide. *J. Adv. Nurs.* **72**, 2954–2965 (2016)
20. Guest, G., MacQueen, K.M., Namey, E.E.: Applied Thematic Analysis. Sage Publications (2012)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



# Author Index

## A

Aldaej, Abdullah 67  
Anslow, Craig 50

## B

Bablo, Jędrzej 34  
Barbala, Astri 165  
Barroca, Leonor 115  
Becker, Steffen 3  
Biddle, Robert 50  
Breitenbücher, Uwe 3

## C

Conboy, Kieran 20

## D

Deshpande, Advait 115  
Dey, Tapajit 20

## F

Fitzgerald, Brian 20

## G

Gregory, Peggy 115  
Guerra, Eduardo 77  
Gupta, Varun 67

## H

Healy, Robert 20  
Hyrynsalmi, Sami 97

## K

Klotins, Eriks 149  
Krieger, Niklas 3  
Křivánková, Michaela 183  
Kropp, Martin 50

## M

Marcinkowski, Bartosz 34  
Melegati, Jorge 77  
Moe, Nils Brede 132

## N

Nguyen-Duc, Anh 67  
Niva, Piret 97

## P

Paasivaara, Maria 97  
Papadeas, Stavros 115  
Przybyłek, Adam 34

## R

Remta, Daniel 183

## S

Sallin, Marc 50  
Sharp, Helen 115  
Smite, Darja 132  
Speth, Sandro 3  
Sporsem, Tor 165  
Stray, Viktoria 165

## T

Talbert-Goldstein, Elliot 149

## V

Viviani, Luiz 77

## W

Wang, Xiaofeng 77  
Wippermann, Pia 3