

**Coordination Mechanisms in Large-Scale Agile Software
Development:
A Longitudinal Empirical Investigation**

Marthe Nordengen Berntzen

Thesis submitted for partial fulfillment of
the requirements for the degree of Philosophiae Doctor

Department of Informatics
Faculty of Mathematics and Natural Sciences University of Oslo

May 2023



Contents

- ABSTRACT VII**
- LIST OF PAPERS..... IX**
- ACKNOWLEDGEMENTS XI**
- LIST OF TABLES..... XIII**
- LIST OF FIGURES..... XIV**
- PART I: SUMMARY 15**
- 1 INTRODUCTION 15**
 - 1.1 RESEARCH QUESTIONS..... 16
 - 1.2 RESEARCH SETTING 17
 - 1.3 THESIS STATEMENT AND OBJECTIVE..... 18
 - 1.4 CONTRIBUTIONS 18
 - 1.5 THESIS STRUCTURE..... 19
- 2 BACKGROUND..... 23**
 - 2.1 LARGE-SCALE AGILE SOFTWARE DEVELOPMENT 23
 - 2.1.1 *Characteristics of large-scale agile*..... 23
 - 2.1.2 *Challenges with large-scale agile software development*..... 24
 - 2.1.3 *Large-scale agile frameworks* 25
 - 2.2 THEORETICAL APPROACHES TO COORDINATION 26
 - 2.2.1 *Malone and Crowston’s Coordination Theory*..... 26
 - 2.2.2 *Relational Coordination Theory*..... 28
 - 2.2.3 *The theory of coordination in agile software development* 29
 - 2.2.4 *Creating a coordinating mechanism in practice* 30
 - 2.2.5 *Other theoretical approaches* 31
 - 2.2.6 *The taxonomy of dependencies in agile software development* 33
- 3 RESEARCH METHODS..... 35**
 - 3.1 RESEARCH CONTEXT 35
 - 3.1.1 *Research perspective* 35
 - 3.1.2 *Research approach* 36
 - 3.2 DATA COLLECTION 39
 - 3.2.1 *Observations* 39
 - 3.2.2 *Interviews*..... 40
 - 3.2.3 *Supplementary documentation*..... 41
 - 3.3 THE RESEARCH CASE 42
 - 3.3.1 *The case organization*..... 42
 - 3.3.2 *The software products*..... 43
 - 3.3.3 *The team organization and the large-scale agile environment* 45
 - 3.4 DATA ANALYSIS..... 47
 - 3.4.1 *Thematic analysis* 47
- 4 FINDINGS..... 53**
 - 4.1 RQ1: WHICH COORDINATION MECHANISMS ARE USED TO MANAGE INTER-TEAM DEPENDENCIES IN LARGE-SCALE AGILE SOFTWARE DEVELOPMENT?..... 53
 - 4.1.1 *Coordination meetings*..... 54
 - 4.1.2 *Coordination roles*..... 59
 - 4.1.3 *Coordination tools and artifacts*..... 62

4.1.4 <i>New insights on previous analyses</i>	65
4.2 RQ2: HOW ARE COORDINATION MECHANISMS USED TO MANAGE DEPENDENCIES IN LARGE-SCALE AGILE SOFTWARE DEVELOPMENT?	69
4.2.1 <i>The TOPS framework</i>	69
4.2.2 <i>The TOPS characteristics of coordination meetings</i>	70
4.2.3 <i>The TOPS characteristics of coordination roles</i>	71
4.2.4 <i>The TOPS characteristics of coordination tools and artifacts</i>	72
4.2.5 <i>Three detailed examples on how coordination mechanisms are used to manage dependencies.</i> 72	
4.3 RQ3: HOW CAN WE ANALYZE COORDINATION MECHANISMS IN LARGE-SCALE AGILE?	75
4.3.1 <i>Applying FALC</i>	78
4.3.2 <i>A flexible and adaptable framework</i>	80
5 DISCUSSION	82
5.1 A COMPREHENSIVE FRAMEWORK FOR A MATURING FIELD	82
5.2 TOWARDS A THEORY OF COORDINATION MECHANISMS IN LARGE-SCALE AGILE	83
5.3 AN ACTIONABLE CONCEPTUALIZATION OF COORDINATION MECHANISMS IN LARGE-SCALE AGILE.	84
5.4. IMPLICATIONS FOR PRACTICE	86
5.5 FUTURE WORK	87
5.6 REFLECTIONS ON THE RESEARCH PROCESS	88
5.6.1 <i>The role of the researcher</i>	88
5.6.2 <i>Ethical considerations</i>	90
5.6.3 <i>Limitations</i>	90
6 CONCLUSION	93
REFERENCES	95
APPENDICES	105
APPENDIX A: PARTICIPANT CONSENT FORM.....	105
APPENDIX B: OVERVIEW OF THE MEETING OBSERVATIONS.....	109
APPENDIX C: OVERVIEW OF THE INTERVIEWS.....	113
PAPER 1: THE PRODUCT OWNER IN LARGE-SCALE AGILE: AN EMPIRICAL STUDY THROUGH THE LENS OF RELATIONAL COORDINATION THEORY	115
ABSTRACT	115
1 INTRODUCTION	115
2 BACKGROUND AND RELATED WORK.....	116
2.1 <i>The Product Owner role in large-Scale Agile</i>	116
2.2 <i>Relational Coordination Theory</i>	117
3 METHOD	118
3.1 <i>Case description</i>	119
3.2 <i>Data collection</i>	119
3.3 <i>Data analysis</i>	120
4 RESULTS	120
4.1 <i>Coordination between POs</i>	120
4.2 <i>Coordination between POs and their teams</i>	123
5 DISCUSSION AND CONCLUSION	124
5.1 <i>Coordination practices varies between the POs</i>	125
5.2 <i>Changes in coordination over time</i>	125
5.3 <i>Unscheduled and frequent coordination enables high-quality communication</i>	125
5.4 <i>Implications for theory and future research</i>	126
5.5 <i>Implication for practice</i>	126
5.6 <i>Limitations and concluding remarks</i>	127
REFERENCES.....	128

PAPER 2: COORDINATION STRATEGIES: MANAGING INTER-TEAM COORDINATION CHALLENGES IN LARGE-SCALE AGILE	131
ABSTRACT	131
1 INTRODUCTION	131
2 BACKGROUND AND RELATED WORK	133
2.1 <i>Managing Dependencies in Large-scale Agile Development</i>	133
2.2 <i>Coordination Strategies</i>	133
3 METHOD AND ANALYSIS	135
3.1 <i>Case Description</i>	135
3.2 <i>Data Collection and Analytical Procedures</i>	136
3.3 <i>Limitations and Threats to Validity</i>	137
4 FINDINGS	137
4.1 <i>Strategy 1: Aligning Autonomous Teams</i>	137
4.2 <i>Strategy 2: Gaining and Maintaining Overview Across Teams</i>	139
4.3 <i>Managing Prioritization Issues</i>	140
4.4 <i>Managing Architecture and Technical Dependencies</i>	141
5 DISCUSSION	142
5.1 <i>Implications for Practice</i>	144
6 CONCLUSION AND FUTURE RESEARCH.....	144
REFERENCES.....	145
PAPER 3: A TAXONOMY OF INTER-TEAM COORDINATION MECHANISMS IN LARGE-SCALE AGILE	147
ABSTRACT	147
1 INTRODUCTION	148
2 BACKGROUND AND RELATED WORK	149
2.1 <i>Agile Software Development at Scale</i>	149
2.2 <i>Perspectives on Coordination and Coordination Mechanisms</i>	150
2.3 <i>Coordination Challenges in Large-Scale Agile</i>	151
2.4 <i>Inter-team Coordination Mechanisms in Large-scale Agile</i>	152
3 RESEARCH DESIGN	153
3.1 <i>Case Description</i>	154
3.2 <i>Data Collection</i>	156
3.3 <i>Data Analysis</i>	159
4 CASE STUDY RESULTS.....	162
4.1 <i>Inter-team Coordination Meetings</i>	162
4.2 <i>Inter-team Coordination Roles</i>	166
4.3 <i>Inter-team Coordination Tools and Artefacts</i>	168
5 DISCUSSION	169
5.1 <i>A Taxonomy of Inter-Team Coordination Mechanisms</i>	170
5.2 <i>Extending the Socio-Technical Perspective</i>	173
5.3 <i>IMPLICATIONS FOR PRACTICE</i>	176
6 LIMITATIONS AND EVALUATION.....	176
7 CONCLUSIONS.....	178
REFERENCES.....	179
PAPER 4: RESPONDING TO CHANGE OVER TIME: A LONGITUDINAL CASE STUDY ON CHANGES IN COORDINATION MECHANISMS IN LARGE-SCALE AGILE	185
ABSTRACT	185
1 INTRODUCTION	186
2 BACKGROUND	188
2.1 <i>Change in Large-Scale Agile Software Development</i>	188
2.2 <i>Coordination and Coordination Mechanisms in Large-Scale Agile</i>	189

2.3. <i>A Process-theoretical Approach to Change in Large-scale Agile</i>	191
3 RESEARCH METHODS	192
3.1 <i>Case Description</i>	193
3.2 <i>Data Collection</i>	194
3.3 <i>Data Analysis</i>	197
4 FINDINGS	199
4.1 <i>External Change Events</i>	200
4.2 <i>Internal Change Events</i>	202
4.3 <i>Continuous Changes in Coordination Mechanisms</i>	205
5 DISCUSSION	209
5.1 <i>External and Internal Drivers of Change in Coordination Mechanisms</i>	209
5.2 <i>Continuous Growth Requires Continuous Change and Improvement</i>	212
5.3 <i>Responding to Change by Using the Right Mechanisms at the Right Time</i>	213
5.4 <i>Implications for Theory</i>	214
5.5 <i>Implications for Practice</i>	214
5.6 <i>Evaluation of Limitations</i>	216
6 CONCLUSIONS.....	217
APPENDIX A.....	218
APPENDIX B	219
REFERENCES.....	220

Abstract

Managing dependencies is crucial for successful large-scale agile software development. Agile methods emphasize autonomous teams, decentralized decision-making, and continuous learning and improvement. While agile methods enable fast software delivery through incremental and iterative development and frequent deployment, from a large-scale perspective, the high number of roles, teams, practices, tools, and technologies involved, and the size of the system under development, lead to many and often complex dependencies. These dependencies are related to both the structure of the organization and the structure of the system under development. If large-scale agile development initiatives are to succeed, these dependencies must be sufficiently managed. This management of dependencies is achieved through coordination. This Ph.D. thesis focuses on the mechanisms used to manage dependencies in large-scale agile.

Software engineering research has focused on identifying coordination mechanisms and describing how they support coordination, often focusing on changes in coordination following a large-scale agile transformation or the implementation of a large-scale framework. However, we lack a structured yet context-sensitive and adaptable approach to analyzing coordination mechanisms to understand how they support managing dependencies in large-scale agile software development. With this Ph.D. thesis, I seek to fill this gap and provide an actionable approach to advance software engineering research and practice.

The empirical research was conducted as a single-case, longitudinal case study using an ethnographic approach to the data collection. The case organization was Entur, a large-scale public-sector software organization that works with developing a public transportation system using agile methods. To collect the data, I conducted a 1.5 yearlong fieldwork, including 108 meeting observations over 62 days on-site and 37 interviews. I also collected supplementary material such as Slack logs, and Confluence and JIRA documentation. The data material was analyzed using thematic analysis.

In this thesis, I investigate three research questions:

RQ1: *Which coordination mechanisms are used to manage inter-team dependencies in large-scale agile software development?*

RQ2: *How are coordination mechanisms used to manage dependencies in large-scale agile software development?*

RQ3: *How can we analyze coordination mechanisms in large-scale agile software development?*

In response to the first research question, I identified 47 coordination mechanisms used in the large-scale agile program. These were categorized using the taxonomy of inter-team coordination mechanisms, which was developed in research Paper 3. Of the 47 mechanisms, 21 were coordination meetings, for example inter-team retrospectives and stand-up meetings; thirteen were coordination roles, such as product owners, customer managers and program architects; and thirteen were coordination tools and artifacts, such as communication and documentation tools, and inter-team product backlogs. The findings section presents an overview of each of these 47 coordination mechanisms, a description of use, which dependencies they manage, and a categorization of their technical, organizational, physical, and social characteristics (TOPS).

In response to the second research question, I analyzed how each of these 47 mechanisms contributes to managing dependencies using the TOPS framework developed in Paper 3. Additionally, I provide detailed examples on how coordination mechanisms are used to manage dependencies in large-scale agile, both in light of their TOPS characteristics as well as in terms of how they support shared goals, shared knowledge and high-quality coordination and how they can be part of coordination strategies in large-scale agile.

In response to the third research question, I introduce the Framework for Analyzing Large-scale agile Coordination mechanisms (FALC). FALC consists of several elements developed throughout Ph.D. research and presented in the different research papers. The framework consists of four steps for analyzing coordination mechanisms in agile: 1) identifying coordination mechanisms, 2) mapping the mechanisms' TOPS characteristics, 3) map how each mechanism manages dependencies, and 4) analyze change in the environment and the mechanisms themselves. Based on the information gained from these steps, coordination strategies can be formed. The framework is context-sensitive and can be used by researchers and practitioners alike.

Following the answers to each of the three research questions, I discuss the implications and contributions of the research in light of the existing research literature, as well as outlining directions for future research on evaluating FALC. I end by presenting a reflection on the research process and my role as a researcher in a long-term field research project of this scope and magnitude, also pointing to the limitations of the research conducted. It is my hope that the findings of this thesis contribute to research on coordination in large-scale agile and serve to guide practitioners seeking to understand and improve their coordination practices.

List of Papers

The following papers are included in this thesis:

- 1. The Product Owner in Large-Scale Agile: An Empirical Study Through the Lens of Relational Coordination Theory**
Marthe Berntzen, Nils Brede Moe, and Viktoria Stray
In Proceedings of the International Conference on Agile Software Development (pp. 121-136). Springer, Cham, 2019
- 2. Coordination Strategies: Managing Inter-team Coordination Challenges in Large-Scale Agile**
Marthe Berntzen, Viktoria Stray, and Nils Brede Moe
In Proceedings of the International Conference on Agile Software Development (pp. 140-156). Springer, Cham, 2021
- 3. A Taxonomy of Inter-Team Coordination Mechanisms in Large-Scale Agile**
Marthe Berntzen, Rashina Hoda, Nils Brede Moe, and Viktoria Stray
IEEE Transactions on Software Engineering, 49, 2 (pp. 699-718), 2022
- 4. Responding to Change Over Time: A Longitudinal Case Study on Changes in Coordination Mechanisms in Large-Scale Agile**
Marthe Berntzen, Viktoria Stray, Nils Brede Moe and Rashina Hoda
Accepted for publication in Empirical Software Engineering

My contributions

As the first author, I was responsible for analyzing and writing all four papers. My degree of independence as first author increased paper by paper. I collected almost all the data except for two interviews, which were held by Dr. Viktoria Stray. I wrote all field notes, transcribed all the interviews, and processed all raw data before conducting all analyses. However, all co-authors contributed with input and feedback during regular discussions. In Paper 1, Dr. Nils Brede Moe and Dr. Stray contributed to the analyses in a joint analysis workshop and the writing process. In Paper 2, Dr. Stray and Dr. Moe supported the analytical process through ongoing discussions. They also contributed to the writing process. For Paper 3, Dr. Rashina Hoda contributed greatly to the early analyses through a series of discussions. She also continued to contribute to the ongoing analyses and to the writing process. Dr. Moe and Dr. Stray contributed during later phases. For Paper 4, Dr. Stray, Dr. Moe, and Dr. Hoda supported the analytical process through ongoing analytical discussions and by reviewing and commenting on drafts of the paper.

Acknowledgements

As I conclude this Ph.D. project, my gratitude goes to the many people I am indebted to for helping and supporting me throughout these years. This dissertation would not have been completed without the support of my supervisors, colleagues, family, and friends.

First and foremost, my most sincere and heartfelt thanks go out to my supervisor, Associate Professor Viktoria Stray. Thank you for believing in me from the beginning and for always supporting me. Also, thank you for being such a good colleague. I hope our collaboration will continue for years. I also sincerely thank my secondary supervisor, Chief Scientist Nils Brede Moe. Your guidance and input have greatly improved my work. Thank you both for making this Ph.D. fun to work with (much of the time).

I further want to extend my gratitude to Associate Professor Rashina Hoda for welcoming me to Australia and to Monash University, taking the time to collaborate with me both during my stay in 2020 and in the following years. I have learned so much from you.

My thanks go out to my colleagues at the Department of Informatics, the team members at SINTEF Digital, and the participants in the A-team research project. You have all contributed with valuable input and interesting discussions that have helped form this thesis.

I also owe my sincere gratitude to Professor Sut I Wong at BI Norwegian Business School for inspiring me to start my research career, always believing in me, and encouraging me to pursue a Ph.D. I will always cherish our collaboration and friendship.

My gratitude also goes out to all the wonderful people at Entur, the company where I did my fieldwork and collected the data for this thesis. Thank you so much for allowing me to follow you in your day-to-day experiences and for making me always feel welcome. To all the people contributing with interviews, thank you for sharing your insights and reflections.

Finally, some words to my family and friends. My life changed considerably more than once during these years, and your love and support helped me stay on track and keep my eyes on the goal of completing this Ph.D. To my husband Thorvald, I honestly could not have done this without you. My parents, sister, in-laws, and my closest friends, thank you for listening, trying to understand, and always encouraging me to keep going. You all made a difference.

My wonderful daughter, Sigrid, the final words are for you. I hope to teach you that you can do anything you want, but that you never have to do it all by yourself.

Marthe Nordengen Berntzen
Oslo, May 2023

List of tables.

Table 1. The papers and their roles in answering the research questions of the thesis.	17
Table 2. Other papers.....	22
Table 3. Selected definitions of large-scale from the literature.....	23
Table 4. A typology of dependencies and coordination mechanisms.	27
Table 5. Elements of relational coordination.....	28
Table 6. A taxonomy of dependencies for agile software development.	33
Table 7. Data material used for the different papers.	41
Table 8. List of all coordination meetings observed during the fieldwork.....	56
Table 9. List of all coordination roles observed during the fieldwork.	60
Table 10. List of all coordination tools and artifacts observed during the fieldwork.....	63
Table 11. Excluded elements.....	68
Table 12. A practical guideline for using the FALC.	81
Table 13. Mapping FALC against the components of a theory for explaining	84
Table 14. Evaluating FALC against the quality criteria by Nickerson et al. (2013).	88

List of figures.

Figure 1. Creating coordination mechanisms through coordinating.	31
Figure 2. Overview of paradigms, methodologies, and methods in qualitative research.....	36
Figure 3. Data collection timeline	39
Figure 4. A retrospective meeting at Entur.	40
Figure 5. A scheduled meeting at Entur.	41
Figure 6. Enturs internal and external environment.	43
Figure 7. Screenshots from the Entur mobile application (14.02.23).	44
Figure 8. The early team organization (approx. 2016-2017).	45
Figure 9. Enturs matrix team organization 2018-2020.....	46
Figure 10. Enturs team organization in service areas from around 2020.....	47
Figure 11. Overview of the six phases of thematic analysis.	49
Figure 12. Illustration of the thematic analysis for Paper 4.	51
Figure 13. Distribution of organizational level of the coordination mechanisms.	53
Figure 14. The taxonomy of inter-team coordination mechanisms in large-scale agile	54
Figure 15. A team-level role operates as an inter-team coordination mechanism.	65
Figure 16. Examples of categorizations of coordination mechanisms.....	66
Figure 17. The TOPS framework.	69
Figure 18. Meeting invitation to the tech lead forum (translated from Norwegian).	73
Figure 19. A screenshot of a Slack channel.	75
Figure 20. The Framework for Analyzing Large-scale agile Coordination mechanisms	77
Figure 21. The TOPS visual template (From Paper 3).....	78
Figure 22. A model of change in coordination mechanisms over time.....	79

PART I: Summary

1 Introduction

During the past two decades, agile methods have become the dominating approach to software development. Since the Agile manifesto was introduced in 2001 (Fowler & Highsmith, 2001), agile has evolved and expanded. Today, the term ‘agile’ covers a variety of meanings, including a range of methods, practices, tools, and techniques that support the process of collaborative software development. Agile methods were initially intended for single-project teams but have spread to large, multi-team projects and organizations as a whole (Dingsøy et al., 2019; Hoda et al., 2018; Rigby et al., 2018). This dissertation focuses on such large-scale agile development settings. The popularity of agile can be attributed to its many alleged benefits, including improved customer collaboration, faster time to market, improved knowledge sharing within and across teams, increased learning, faster software deployment and delivery, higher employee motivation, and more efficient improvement of processes and practices through continuous improvement, to name a few (Hoda et al., 2018; Palopak et al., 2023).

Despite this popularity, there are challenges associated with large-scale agile, including implementing agile methods at large scale, lack of organizational support and clashes between agile teams and more hierarchical organizational structures (Dikert et al., 2016; Edison et al., 2022; Palopak et al., 2023). These and other challenges stem from the increased complexity that follows large software systems and large development organizations, and the unavoidable coordination required when many teams work together towards a common development goal (Bick et al., 2018; Dikert et al., 2016; Kalenda et al., 2018). Accordingly, coordination in large-scale agile has become an important research topic.

Coordination is an interdisciplinary field of study that span field like organization science, information systems, computer science, psychology and sociology, and software engineering (Dingsøy et al., 2022; Okhuysen & Bechky, 2009). Coordination can be defined as the management of interdependent activities (Malone & Crowston, 1994). In the context of large-

scale agile coordination is needed when, for example, multiple teams need to provide code as input to an overall software product and depend on each other to finalize their work, or when several teams require the knowledge of the same person (such as an architect) at the same time.

Much attention has been devoted to understanding and improving coordination in large-scale agile, not only in research but also in practice by the widespread use of large-scale agile frameworks that aim to support coordination (Edison et al., 2022). Despite such efforts, coordination remains the main challenge in large-scale agile (Bass & Salameh, 2020; Palopak et al., 2023).

1.1. Research questions

There is a recognized need for more knowledge on coordination in large-scale agile software development settings (Bass & Salameh, 2020; Edison et al., 2022; Uludağ et al., 2022). Specifically, there is a need to understand which practices and tools (i.e., mechanisms) contribute to managing dependent activities to achieve effective coordination (Dingsøyr et al., 2017; Edison et al., 2022). In this Ph.D. thesis, I therefore aim to contribute to a better understanding of how coordination mechanisms can be used to support coordination in large-scale agile software development projects. Specifically, I investigate three research questions:

RQ1: Which coordination mechanisms are used to manage inter-team dependencies in large-scale agile software development?

RQ2: How are coordination mechanisms used to manage dependencies in large-scale agile software development?

RQ3: How can we analyze coordination mechanisms in large-scale agile software development?

These research questions are open and descriptive and therefore suitable for exploration by qualitative methodologies (Runeson & Höst, 2009; Sharp et al., 2016). The contributions of each paper to the three research questions are provided in Table 1.

Table 1. The papers and their roles in answering the research questions of the thesis.

Paper number	Role in answering the research question(s)
Paper 1	Contributes to RQ1 and RQ2 by reporting on how product owners used 10 coordination mechanisms (RQ1) for within-team and inter-team coordination, focusing on how the mechanisms contribute to shared knowledge and shared goals as well as the communication quality among product owners (RQ2).
Paper 2	Contributes to RQ1 and RQ2 by reporting on how 19 coordination mechanisms (RQ1) support dependency management across four coordination strategies, including 1) aligning autonomous teams, 2) maintaining overview in the large-scale setting, 3) managing prioritizations, and 4) managing architecture and technical dependencies (RQ2).
Paper 3	Contributes to all three research questions by reporting 27 inter-team coordination mechanisms (RQ1) and presents a taxonomy of inter-team coordination mechanism and an analysis of their technical, organizational, physical, and social characteristics, and how they are used to manage dependencies (RQ2 and RQ3).
Paper 4	Contributes to all research questions, by identifying how external and internal events, as well as a day-to-day focus on continuous improvement, are part of shaping which coordination mechanisms (RQ1) are used, and also how they are used, to manage dependencies over time (RQ2). From this, we propose a practical approach to study change in coordination mechanisms (RQ3).

1.2 Research setting

I conducted fieldwork from September 2018 – January 2020 in Entur, a public sector organization working with developing and maintaining a digital national travel planner. Access to the organization was provided via my supervisors, who had collaborated with Entur in the Norwegian Research Council project ‘A-team’. However, maintaining access has primarily been my responsibility, including providing feedback to the organization (Crang & Cook, 2007; Walsham, 2006, 2012). I was given access to the organization during initial meetings with Entur representatives. We agreed that I would spend considerable time on-site to get a detailed understanding of their daily work. I was given a key card, a place to sit while working, and access to many of their digital systems, including the collaboration tool Slack.

The case was chosen because of the many interesting features relevant to the study of inter-team coordination. At Entur, the software development organization was arranged into development teams. They had used agile methods since the outset in 2016 and were considered a modern development company. The teams depended on each other to various degrees in solving their tasks. To coordinate, they used digital tools, physical artifacts, and various forms of meetings. In terms of agile methods, the teams relied on agile methods of choice, and they could choose how they solved their tasks. The organization had a complex external environment, with stakeholders from governmental agencies to public transportation operators and end-users, all with different needs, expectations, and timeframes. The organization had already grown fast, from five teams when they were established in 2016 to 13 teams when my fieldwork began in 2018. During the 1.5 years I followed them, they grew to 17 permanent development teams as well as various support teams and temporary teams, and the growth has continued. All in all, these and other conditions created a challenging environment for inter-team coordination, which made an interesting case for exploring my research question.

1.3 Thesis statement and objective

The thesis statement is as follows:

In large-scale agile software development, dependencies within and between teams must be managed by using coordination mechanisms.

A better understanding of which coordination mechanisms can be used to manage dependencies and how these mechanisms may change in response to a complex large-scale environment will contribute to improving coordination, thereby facilitating more successful software development and continuous value delivery.

Based on this statement, the objective of this Ph.D. thesis is to collect evidence about which coordination mechanisms are used, in what way, to manage dependencies, and to synthesize this evidence to advance knowledge on coordination in large-scale agile.

1.4 Contributions

To achieve the objective of the thesis, I have conducted a longitudinal field study using ethnographic methods, where I followed a large-scale agile development program for 1.5 years. The data has been analyzed several times through thematic analysis.

The following contributions are made:

- A comprehensive collection of coordination mechanisms used over 1.5 years in a large-scale agile development program.
- A rich and detailed description of the empirical and analytical work, which contributes to the usefulness and trustworthiness of the findings and conclusions drawn from this research.
- A thorough investigation of how coordination mechanisms are used in practice in large-scale agile software development.
- The main contribution of this thesis is the **Framework for Analyzing Large-scale agile Coordination mechanisms (FALC)**, consisting of the following elements:
 - A taxonomy of inter-team coordination mechanisms which can provide structure for future empirical and analytical work on the research topic.
 - A framework for analyzing the technical, organizational, physical, and social (TOPS) characteristics of coordination mechanisms, which allows for a deeper understanding of how individual mechanisms contribute to managing dependencies.
 - A practical approach to analyzing change in coordination mechanisms.

From a research perspective, FALC can be used as part of studies addressing coordination in large-scale agile, growing the knowledge base of which mechanisms are used in what way to improve coordination and, ultimately, supporting continuous software delivery in large-scale

agile. From a practical perspective, the framework can be used to analyze and understand coordination and serve as input for companies to form their own coordination strategies.

By this, the Ph.D. thesis advances knowledge on coordination mechanisms in large-scale agile, with the goal of assisting future research and practice in improving dependency management in large-scale agile software development. The knowledge derived is hoped to contribute to successful software development and continuous value delivery using agile methods in large-scale settings.

1.5 Thesis structure

This thesis contains a summary and four research papers.

Summary: The summary consists of six chapters. Chapter 1 introduces the thesis research topic, organization, and Chapter 2 presents the research background. In Chapter 3, the research methods and methodological considerations for this thesis are presented. Chapter 4 presents the results of the synthesis, which are further discussed in Chapter 5. Finally, Chapter 6 contains concluding remarks. At the end of the summary, three appendices provide additional details.

Papers: The abstracts of the four papers included in this thesis is presented below.

Paper 1, “*The Product Owner in Large-Scale Agile: An Empirical Study Through the Lens of Relational Coordination Theory*” (Berntzen et al., 2019) examines how product owners coordinate through a theoretical framework focusing on the importance of *shared goals*, *shared knowledge*, and *high-quality communication* to achieve efficient coordination. Additionally, the study reports on 10 coordination mechanisms product owners use to coordinate within and across teams.

Abstract: In agile software development, a core responsibility of the product owner (PO) is to communicate business needs to the development team. In large-scale agile software development projects, many teams work toward an overall outcome, but they also need to manage interdependencies and coordinate efficiently. In such settings, POs need to coordinate knowledge about project status and goal attainment both within and across the development teams. Previous research has shown that the PO assumes a wide set of roles. Still, our knowledge about how POs coordinate amongst themselves and with their teams in large-scale agile is limited. In this case study, we explore PO coordination in a large-scale development program through the theoretical lens of Relational Coordination Theory. Our findings suggest that 1) coordination varies depending on the context of each PO, 2) a focus on achieving high-quality communication changes coordination over time, and 3) unscheduled coordination enables high-quality communication.

Paper 2, “*Coordination Strategies: Managing Inter-team Coordination Challenges in Large-Scale Agile*” (Berntzen et al., 2021), examines how coordination mechanisms are used together to resolve coordination challenges. The study builds on research conducted within agile development teams, where one coordination strategy was proposed. Extending this work to the inter-team level, four strategies across 19 coordination mechanisms were identified.

Abstract: Inter-team dependency management in large-scale software development can be challenging when relying on agile development methods that emphasize iterative and frequent delivery in autonomous teams. Previous research has introduced the concept of coordination strategies, which refer to as set of coordination mechanisms to manage dependencies. We report on a case study in a large-scale agile development program with 16 development teams. Through interviews, meeting observations, and supplemental document analyses, we explore the challenges to inter-team coordination and how dependencies are managed. We found four coordination strategies: 1) aligning autonomous teams, 2) maintaining overview in the large-scale setting, 3) managing prioritizations, and 4) managing architecture and technical dependencies. This study extends previous research on coordination strategies within teams to the inter-team level. We propose that large-scale organizations can use coordination strategies to understand how they coordinate across teams and manage their unique coordination situation.

Paper 3, “*A Taxonomy of Inter-Team Coordination Mechanisms in Large-Scale Agile*” (Berntzen et al., 2022), describes 27 coordination mechanisms used across teams over 1.5 years in a large-scale agile organization. In this study, a taxonomy to systematize coordination mechanisms, as well as a novel framework for studying the underlying characteristics of coordination mechanisms is developed.

Abstract: In large-scale agile software development, many teams work together to achieve overarching project goals. The more teams, the greater the coordination requirements. Despite the growing popularity of large-scale agile, inter-team coordination is challenging to practice and research. We conducted a case study over 1.5 years in a large-scale software development firm to better understand which inter-team coordination mechanisms are used in large-scale agile and how they support inter-team coordination. Based on a thematic analysis of 31 interviews, 113 hours of observations, and supplemental material, we identified 27 inter-team coordination mechanisms. From this, we offer the following contributions. First, we propose a taxonomy of inter-team coordination with three categories: coordination meetings, such as communities of practice, inter-team stand-ups, and retrospectives; coordination roles such as the program architects and the platform team; and coordination tools and artifacts, such as Slack and JIRA as well as inter-team task boards, product backlogs, and roadmaps. Second, the coordination mechanisms displayed combinations of four key characteristics, technical, organizational, physical, and social (TOPS), which form the basis of the TOPS framework to capture the multifaceted characteristics of coordination mechanisms. *Technical* relates to the software product and/or technical tools supporting software development. *Organizational* pertains to the structural aspects of the organization. *Physical* refers to tangible or spatial characteristics. *Social*

captures interpersonal and community-based characteristics. Finally, the taxonomy and the TOPS framework provide a knowledge base and a structured approach for researchers to study as well as for software practitioners to understand and improve inter-team coordination in large-scale agile.

Paper 4, “*Responding to Change Over Time: A Longitudinal Case Study on Changes in Coordination Mechanisms in Large-Scale Agile*” (Berntzen et al., in press) explores how coordination mechanism change over time. This paper takes a broader approach, in exploring how both external and internal change events and continuous improvement influence changes in coordination mechanisms over time. In the paper, I also suggest how large-scale agile organizations may use this insight to respond to change in a timely and agile manner.

Abstract:

Context: Responding to change and continuously improving processes, practices, and products are core to agile software development. It is no different in large-scale agile, where multiple software development teams need to respond both to changes in their external environments and to changes within the organization.

Objective: With this study, we aim to advance knowledge on coordination in large-scale agile by developing a model of the types of organizational changes that influence coordination mechanisms.

Method: We conducted a longitudinal case study in a growing large-scale agile organization, focusing on how external and internal changes impact coordination over time. We collected our data through 62 days of fieldwork across one and a half years. We conducted 37 interviews, observed 108 meetings at all organizational levels, collected supplementary material such as chat logs and presentations, and analyzed the data using thematic analysis.

Results: Our findings demonstrate how external events, such as onboarding new clients, and internal events, such as changes in the team organization, influence coordination mechanisms in the large-scale software development program. We find that external and internal change events lead to the introduction of new coordination mechanisms, or the adjustment of existing ones. Further, we find that continuous scaling requires continuous change and adjustment. Finally, we find that having the right mechanisms in place at the right time strengthens resilience and the ability to cope with change in coordination needs in complex large-scale environments.

Conclusions: Our findings are summarized in an empirically based model that provides a practical approach to analyzing change, aimed at supporting both researchers and practitioners dealing with change in coordination mechanisms in large-scale agile development contexts.

In addition to the papers included in this thesis, I have contributed to several papers on topics relevant to large-scale agile software development and coordination. I chose not to include them in the dissertation because they are not based on the fieldwork I conducted for this thesis, and 1) because I was not the lead author and 2) I was the lead author, but the topic was not inter-team coordination in large-scale agile. These papers are listed in Table 2 for reference.

Table 2. Other papers.

	Authors, title, outlet, year
1.	S. I. Wong, M. Berntzen, G. Warner-Söderholm, and S. R. Giessner, "The negative impact of individual perceived isolation in distributed teams and its possible remedies," <i>Human Resource Management Journal</i> , (2022).
2.	T. Gustavsson, M. Berntzen, and V. Stray, "Changes to team autonomy in large-scale software development: a multiple case study of Scaled Agile Framework (SAFe) implementations," <i>International Journal of Information Systems and Project Management</i> , vol. 10, no. 1, pp. 29–46, (2022).
3.	V. Stray, N. B. Moe, H. Vedal, and M. Berntzen, "Using objectives and key results (OKRs) and slack: a case study of coordination in large-scale distributed agile," in <i>Proceedings of the 55th Hawaii International Conference on System Sciences</i> , (2022).
4.	M. Berntzen and S. I. Wong, "Autonomous but Interdependent: The Roles of Initiated and Received Task Interdependence in Distributed Team Coordination," <i>International Journal of Electronic Commerce</i> , vol. 25, no. 1, (2021).
5.	H. Vedal, V. Stray, M. Berntzen, and N. B. Moe. "Managing dependencies in large-scale agile," in <i>Agile Processes in Software Engineering and Extreme Programming – Workshops</i> , (2021), pp. 52-61.
6.	I. Hukkelberg and M. Berntzen, "Exploring the Challenges of Integrating Data Science Roles in Agile Autonomous Teams," in <i>Agile Processes in Software Engineering and Extreme Programming – Workshops</i> , (2019), pp. 37–45.
7.	M. Berntzen and S. I. Wong, "Coordination in Distributed, Self-managing Work Teams: The Roles of Initiated and Received Task Interdependence," in <i>Proceedings of the 52nd Hawaii International Conference on System Sciences</i> , (2019), pp. 973-982.
8.	S. I. Wong and M. Berntzen, "Transformational leadership and leader–member exchange in distributed teams: The roles of electronic dependence and team task interdependence," <i>Computers in Human Behavior</i> , vol. 92, pp. 381–392, (2019).

2 Background

In this chapter, I first introduce large-scale agile software development and present key challenges. Next, I review selected literature on the interdisciplinary topic of coordination from relevant research fields. I end the chapter by presenting the operationalizations of the concepts used in this thesis.

2.1 Large-scale agile software development

This section presents background information on agile and large-scale agile. Agile methods have become the dominant approach to software development over the past two decades. As a result, a large amount of research on agile has been conducted, from agile requirements engineering to agile micro-practices and how to use agile methods in large-scale projects and organizations (Hoda et al., 2018; Kalenda et al., 2018; Rigby et al., 2018). This Ph.D. thesis focuses on ‘large-scale agile’, which, broadly speaking, refers to the use of agile methods and practices across multiple development teams.

2.1.1 Characteristics of large-scale agile

There is no unified definition of ‘large-scale’ or ‘large-scale agile’ in the research literature. Some definitions emphasize the size of the system, or the lines of code written, while others focus on the number of individuals or teams involved in the development process (Edison et al., 2022). Table 3 shows some of the definitions of large-scale agile available in the software engineering field. In this thesis, we have used the definition by Dingsøyr et al. (2014) for Papers 1 and 2 and the definition by Dikert et al. (2016) in Papers 3 and 4.

There are many variations of large-scale agile methods, and there is no unified agreement on which specific methods or practices constitute large-scale agile or if any methods and practices are better than others (Edison et al., 2022; Kalenda et al., 2018). This makes it difficult to provide one general definition, despite calls for a standard understanding of method usage in software engineering research and practice (Dittrich, 2016; Kitchenham et al., 2004; Stol &

Table 3. Selected definitions of large-scale from the software engineering and information systems literatures.

Publication	Definition
Dingsøyr et al. (2014)	Defines ‘large-scale agile development projects’ as consisting of two-nine teams, where coordination can be achieved by one inter-team coordination forum. Further defines ‘very large-scale’ as consisting of more than ten teams, where more than one inter-team coordination forum is needed.
Rolland et al. (2016)	Refers to ‘large-scale’ as a complex integration with various internal and external information systems, and multiple stakeholders with different interests.
Dikert et al. (2016)	Defines ‘large-scale’ as than 50 developers or at least six teams working on a common software product or project.
Uludağ et al. (2022)	Defines ‘large-scale agile’ as the adoption of agile methods in large agile multi-team settings with at least two teams, or the large-scale adoption of agile methods on the organizational level comprising multiple large agile multi-team settings.

Fitzgerald, 2015). In a recent systematic review on large-scale agile method use, Edison et al. (2022) reviewed 191 primary studies across 134 case organizations and reported on the principles, practices, tools, and metrics used across four large-scale method categories: 1) Large-scale frameworks (to be presented below), 2) customized methods, 3) methods with connecting practices, and 4) methods without connecting practices. The findings of their review show, among others, that a characteristic of large-scale organizations is that they adapt or customize their use of agile practices, tools, and metrics.

Typical elements present in a large-scale setting include agile meetings such as stand-up meetings (Stray et al., 2018), sprint planning, and retrospectives (Andriyani et al., 2017; Przybyłek et al., 2022), agile roles such as the Scrum master (Shastri et al., 2021a; Spiegler et al., 2021) and product owner (Bass, 2015; Paasivaara et al., 2012), and agile tools and artifacts such as program backlogs, wiki's (Dingsøyr et al., 2018; Strode et al., 2012). In addition, various other management practices and elements are commonly used (Batra et al., 2010; Bick et al., 2018; Edison et al., 2022), such as the project manager role (Shastri et al., 2021b), the key performance indicator metric, and risk management tools (Batra et al., 2010) and communication tools like Slack (Stray & Moe, 2020), to name a few. Some authors refer to this mixing of agile and non-agile practices as 'hybrid' large-scale agile, or similar terms, to describe situations where agile practices are used alongside more traditional practices (Bick et al., 2018; Edison et al., 2022). Because of the many and varied approaches to large-scale agile that exist, in this thesis, I use the term large-scale agile in a broad sense to refer to the use of agile methods and practices, as well as other management practices, in a multi-team setting where teams need to coordinate to deliver software.

A recent paper by Palopak et al. (2023) examined how research on agile methods has evolved over the past two decades. They found that since 2013, the four most-researched sub-topics are related to large-scale development (Palopak et al., 2023). These include global software development and global software engineering (Giuffrida & Dittrich, 2015; Herbsleb, 2007; Stray & Moe, 2020), tailoring of agile practices (Bass, 2016; Campanelli & Parreiras, 2015) and scaling agile (Jorgensen, 2019; Kalenda et al., 2018). Finally, many studies have focused on challenges and success factors in large-scale organizations (Dikert et al., 2016; Palopak et al., 2023).

2.1.2 Challenges with large-scale agile software development

Despite the popularity of agile, several challenges and barriers are associated with using agile methods in large-scale settings. For instance, during large-scale agile transformations, challenges include, for example, a lack of organizational commitment and difficulties with implementing agile (Dikert et al., 2016). When agile has been implemented, challenges include measuring progress, managing stakeholders, keeping with the agile principles, and striking the right balance between the need for overview and alignment and the need for autonomy (Edison et al., 2022; Kalenda et al., 2018). This is because, although self-management and team autonomy is core to agile (Hoda et al., 2012; Williams & Cockburn, 2003), in large-scale settings, team autonomy must be balanced with the larger organizational structures because of a greater need for coordination and alignment between the software system(s) under development and the organization, (Bick et al., 2018; Conway, 1968; Govers & van

Amelsoort, 2018). Further, complex products with many technical dependencies may require careful management in large systems, particularly those involving tightly coupled teams and architectures (Dingsøy, Moe, et al., 2017; Yang et al., 2016).

Among these and other challenges, coordination between teams seems to have received the greatest research attention during the past decade (Palopak et al., 2023). As will be explored in detail in this thesis, a multi-team large-scale environment comes with inevitable dependencies of various kinds. These dependencies must be managed to avoid inefficient development processes, delays in software deliveries, or even coordination breakdowns (Cataldo & Herbsleb, 2012). As mentioned above, one dependency category is technical dependencies associated with complex systems. However, software development is a socio-technical activity (Hoda, 2021; Storey et al., 2020), which leads to human-related dependencies in addition to technical ones. These include, for example, situations where many individuals or teams depend on the knowledge of a few expert individuals (Sablis et al., 2020), dependencies stemming from a shortage of resources, or dependencies to other parts of the organization (Dingsøy et al., 2018; Strode, 2016; Vedal et al., 2021).

2.1.3 Large-scale agile frameworks

As scaling agile has become common, a range of commercial frameworks have become available to practitioners. These are aimed at supporting large-scale coordination by providing tools and mechanisms to manage dependencies (Conboy & Carroll, 2019; Edison et al., 2022; Uludağ et al., 2022). In this section, I present some of the most well-known frameworks.

The Scaled Agile Framework (SAFe) is comprised of a detailed collection of best practices and procedures for agile development for large organizations and enterprises (Edison et al., 2022; Kalenda et al., 2018). SAFe includes ideas from agile and lean development and systems thinking and aims to provide value by supporting all parts of the organization, from the team level to the program, portfolio, and even value stream level (Kalenda et al., 2018). SAFe is the most-used large-scale agile framework (Digital.ai, 2020, 2021, 2022), and much research has been conducted to understand and evaluate its effectiveness (Hussain, Perera, et al., 2022; Paasivaara, 2017; Paasivaara et al., 2018; Turetken et al., 2017). From a coordination perspective, Gustavsson and colleagues have studied SAFe at the team-level and inter-team level (Gustavsson, 2019; Gustavsson et al., 2022).

Another framework that has received research attention in recent years is the Spotify model (Kniberg & Ivarsson, 2012), and specifically the terms *guilds*, which can be compared to a community of practice (Smite et al., 2019; Wenger et al., 2002), and *squads* (i.e., teams) (Salameh & Bass, 2019). Other well-known large-scale agile frameworks include Scrum-at-scale, Disciplined Agile Delivery, and Large-Scale Scrum (Edison et al., 2022; Uludağ et al., 2022).

A common critique against the large-scale agile frameworks is that they lack context specificity and requires much tailoring to suit the needs of each specific organization (Conboy & Carroll, 2019; Salameh & Bass, 2022). This requires a lot of time and resources, which in turn creates new dependencies (Edison et al., 2022; Gustavsson, 2019; Gustavsson et al., 2022). Another concern with many of these frameworks is that they provide such detailed descriptions and instructions that one might question how ‘agile’ they really are (Conboy & Carroll, 2019;

Edison et al., 2022). For example, in a study of team autonomy across three organizations, Gustavsson et al. (2022) found that team members perceived (lower levels of autonomy and decision-making. In another recent paper, Carroll et al. (2023) examined agile practices in a large international company that implemented the Spotify model during a large-scale agile transformation. They found that a failure to normalize new practices led to the unraveling of the transformation within 18 months.

A response to these critiques is that the frameworks are intended to be tailored. They provide the basic tools and procedures but are meant to be shaped to fit the organization's context (Kalenda et al., 2018). And indeed, most companies do adapt the large-scale frameworks. According to the recent review by Edison and colleagues (2022), 85 out of 134 organizations had adapted and tailored their large-scale agile method use in some way, either by tailoring existing frameworks, developing their own hybrid methods, or using customized frameworks (typically provided by consultancy firms).

Regardless of the approach or method used, the need to coordinate across teams is a key feature of large-scale agile. This key feature will be the focus of the remainder of this thesis.

2.2 Theoretical approaches to coordination

The word 'coordination' has a variety of meanings, depending on the context in which it is used. As a general definition, the verb 'to coordinate' refers to "*the act of organizing people or groups so that they work together properly and well*"¹.

Coordination is a highly multi-disciplinary research topic, and presenting all approaches to coordination is not within the scope of this summary. In the following, I present some influential contributions to the study of coordination relevant to my focus on coordination within and between agile teams.

2.2.1 Malone and Crowston's Coordination Theory

Malone and Crowston (1994) developed an interdisciplinary, broad-based theory of coordination, which is referred to as 'CT' for the purposes of this summary. Within CT, coordination is defined as a process of "managing dependencies between activities" (Malone & Crowston, 1994, p. 90). This definition is the main definition of coordination used in this thesis and across all papers.

Based on ideas from organization theory, management, economics, and computer science, the basic tenet of CT is that complex organizational systems are made up of dependencies (such as shared resources, task interdependencies, simultaneity constraints, and relationships with clients, each with different sub-dependencies), which constrain situational action and, therefore, must be coordinated. Coordination, then, comprises various coordination processes and mechanisms, each addressing one or more dependencies in a situation (Malone & Crowston, 1994). What these processes and mechanisms look like varies with the context.

¹ Retrieved April 24, 2023, from <https://www.merriam-webster.com/dictionary/coordination>

Table 4. A typology of dependencies and coordination mechanisms (from Malone and Crowston, 1994).

Dependency	Examples of Coordination processes and mechanisms
Shared resources	“First come/first serve”, priority order, budgets, managerial decision-making, market-like bidding
Task assignments	(same as for “shared resources”)
Producer/consumer relationships	
Prerequisite constraints	Notification, sequencing, tracking
Transfer	Inventory management (e.g., “just in time”, “economic order quantity”)
Usability	Standardization, ask users, participatory design
Design for manufacturability	Concurrent engineering
Simultaneity constraints	Scheduling, synchronization
Task/subtask	Goal selection, task decomposition

In the context of large-scale agile software development, they can include, for instance, scheduled and unscheduled meetings, artifacts, and physical settings (Nyrud & Stray, 2017; Strode et al., 2012) that manage dependencies. Table 4 provides the original examples of dependencies and coordination mechanisms.

CT provides a theoretical framework and a much-cited definition of coordination, a modeling framework for analyzing coordination in complex processes, and provides a beginning of a typology of dependencies and coordination mechanisms (Howison et al., 2015). CT has been used in software engineering and systems design, where researchers have noted the importance of coordination challenges and the potential for computer systems to help groups and teams collaborate better (Howison et al., 2015). In the context of agile software development, CT has been applied by Strode and colleagues (2012), who used the theory as the basis for their development of a theory of coordination in agile development.

In a ten-year retrospective of CT research, (Howison et al., 2015) encourages future research, for instance, about the generality of coordination mechanisms and more structured approaches to evaluate coordination processes. As will be shown throughout this thesis, I contribute to this call by developing a general framework for describing coordination mechanisms in a general way and a structured approach for analyzing and thereby evaluating coordination mechanisms.

2.2.2 Relational Coordination Theory

Relational Coordination Theory (RCT) originates in the organization studies field from research conducted in the airline industry in the 1990s (Gittell, 2006), where Gittell observed substantial differences between companies in the extent to which the employees shared collective goals and knowledge towards the overall work process and outcome. Today, RCT is an established and empirically validated theory and has been studied in various fields and industries where large-scale projects are common, such as the airplane, health, and education industries (Bolton et al., 2021; Gittell, 2012). It has also been picked up by information systems and software engineering researchers (Bozan, 2017; Claggett & Karahanna, 2018; Sebastian & Bui, 2012; Stray, Moe, Vedal, et al., 2022). RCT is used as the theoretical lens of Paper 1.

RCT emphasizes the importance of high-quality relationships and high-quality communication for effective coordination. *Relational coordination* is defined as “a mutually reinforcing process of interaction between communication and relationships carried out for the purpose of task integration” (Gittell, 2002, p. 301). According to RCT, relationships provide the necessary bandwidth for coordinating work in settings that are highly interdependent, uncertain, and time constrained. Effective coordination in these settings is carried out through relationships of shared goals, shared knowledge, and mutual respect. These relationships can be between individuals, roles, or even departments and organizations. The basic tenet is that a positive relational context enables people to achieve quality outcomes through a well-coordinated process with less wasted effort (Gittell, 2006). Table 5 provides a summary of the key elements of RCT.

According to RCT, shared knowledge, shared goals, and mutual respect should mutually reinforce frequent, accurate, timely, and problem-solving communication (Gittell, 2002, 2006, 2012). This should contribute to the overall quality of the work process coordination. Finally, relation coordination is stronger in more horizontally designed organizational structures (Gittell & Douglass, 2012), which is important to large-scale agile. RCT provides an interesting lens

Table 5. Elements of relational coordination based on the synthesis by Gittell (from Paper 1).

Relational Coordination	Definition	Specific examples
Shared knowledge	Informs participants of how their own and others’ tasks contribute to the overall work process. A shared understanding of the work process and others’ areas of expertise facilitate knowledge coordination.	Knowledge about overall delivery milestones; Knowledge about which team is working on what, when.
Shared goals	Direct the attention and effort of individuals and groups. Transcend functional goals of different work units and enable unified effort towards a collective outcome.	Keeping in mind overall program goals while working on team goals.
Mutual respect	Valuing others’ contributions and consider the impact of their own actions to the work of others.	Consider the impact of one teams’ work on another; Acknowledge differences in priorities; Trust others’ decisions and work.
High-quality communication	Communication that is frequent, accurate, timely and problem-solving in nature.	Keep meetings relevant, send and receive information at the right time with the right content; constructive feedback

for studying coordination in large-scale agile development because large-scale agile software development processes are also typically characterized by high levels of interdependence, uncertainty, and time pressure and because autonomy is central to agile (Dikert et al., 2016).

2.2.3 The theory of coordination in agile software development

To further advance theory and research on coordination in agile, Strode and colleagues (2012) proposed a theory of coordination in agile development. The theory posits that effective coordination in agile settings is comprised of coordination strategies that contribute to coordination effectiveness (Strode et al., 2012). Coordination strategies are defined as a group of coordination mechanisms that manage dependencies in a situation. They consist of three components; synchronization, structure, and boundary-spanning activities and artifacts (Strode et al., 2012).

The first component, *synchronization*, consists of activities performed by all team members simultaneously and promotes a common understanding of the task, process, and/or expertise of other team members. An example of a synchronization activity can be a meeting, such as the Daily Stand-up Meeting (Stray et al., 2016). Synchronization artifacts are generated during synchronization activities and can be readily visible to the whole agile team, such as a task board, or largely invisible but available as needed, such as a task backlog. The second component, *structure*, relates to coordination mechanisms that have structural qualities in terms of the arrangement of, and relations between, the parts of a complex whole. Three sub-mechanisms are identified under structure: the proximity, availability, and substitutability of team members (Strode et al., 2012). Finally, the third component relates to *boundary-spanning* coordination mechanisms. These may be artifacts and activities related to something external to the agile team. A boundary-spanning mechanism may also be a *coordinator role*, which can be a role taken by a project team member specifically taken to support interaction with people external to the agile development team, such as the product owner (Bass, 2015a; Paasivaara et al., 2012).

Coordination effectiveness, in turn, consists of explicit and implicit effectiveness. Explicit coordination effectiveness occurs when the physical objects (both persons and artifacts) involved in the project are in the right place, at the right time, and in the right state so that they are “ready for use” as perceived by each person involved in the project (Malone & Crowston, 1994; Strode et al., 2012). Having the right tools in place to conduct a video meeting in a distributed project or having available developers to take on new tasks as they flow from a different team are examples of this type of explicit coordination effectiveness. Implicit coordination effectiveness, on the other hand, relates to coordination that occurs within work groups without the explicit passing of messages. It consists of five components; “knowing why,” “knowing what is going on and when,” “knowing what to do and when,” “knowing who is doing what,” and “knowing who knows what” (Faraj & Sproull, 2000; Strode et al., 2012). In other words, implicit coordination requires a high degree of shared goals and an understanding of both one’s own and others’ knowledge. In Paper 2, we applied the theory of coordination in agile development teams at the inter-team level and extended the coordination strategy concept to include inter-team mechanisms.

2.2.4 Creating a coordinating mechanism in practice

Common to the three approaches described so far is that they focus on ‘coordination,’ either as an activity or as a process, rather than on the mechanisms that enable this coordination activity or process.

In their 2012 paper, Jarzabkowski and colleagues argue that there is a tension in the coordination literature in that many approaches, coordination mechanisms are described in a relatively static way as rules and procedures that are applied in response to ‘fixed’ coordination problems and known dependencies (Jarzabkowski et al., 2012). At the same time, empirical research repeatedly demonstrates that reality is much more unpredictable, particularly in high-tech contexts with complex systems and highly knowledge-based work (Faraj & Yan, 2006; Jarzabkowski et al., 2012). Instead, researchers on coordination need to recognize that organizational actors actively create and shape coordination mechanisms through a process of *coordinating* (Jarzabkowski et al., 2012, my italics). From this, building on practice theories of the relationship between structure and action (e.g., Bourdieu, 1977; Giddens, 1993; Orlikowski, 1992), Jarzabkowski et al. (2012) propose a theoretical model to explain how coordinating mechanisms are created in practice through five cycles. In Paper 4, I use this theoretical lens to study changes in coordination mechanisms over time. I have not, however, adopted the term ‘coordinating’ term, to keep in line with the software engineering literature on coordination. In the following, I briefly describe the five-cycle model. Figure 1 presents an illustration adapted from Jarzabkowski et al. (2012).

First, the process starts with some event that *disrupts existing ways of coordinating*. This may be, for example, a reorganization (as in the original paper), a merger, or replacing existing technologies or information systems used. Such an event will substantially alter how people coordinate, leading to a disruption that builds barriers to existing ways of coordinating and broken coordination patterns. In the second cycle, as people make efforts to coordinate and fail to do so effectively, they start to identify patterns of what can no longer be done, leading them to *orient to absences in coordinating*. For example, actors may realize that a meeting is no longer effective or that existing communication channels no longer work optimally. Third, new *elements of coordinating* are created as people try to fill the absences by forming new elements of coordinated activities. Fourth, *new patterns of coordinating* are formed as new coordination mechanisms emerge, and the links between the elements are made visible. In the fifth and final cycle, the new mechanisms are formalized, and new coordination *patterns are stabilized* as people settle with the new ways of coordinating (Jarzabkowski et al., 2012).

This theoretical model provides an alternative way of studying coordination, focusing on the coordination mechanisms as dynamic and changeable entities, which is highly suitable for studying coordination in the often complex and knowledge-intensive settings that make up large-scale agile.

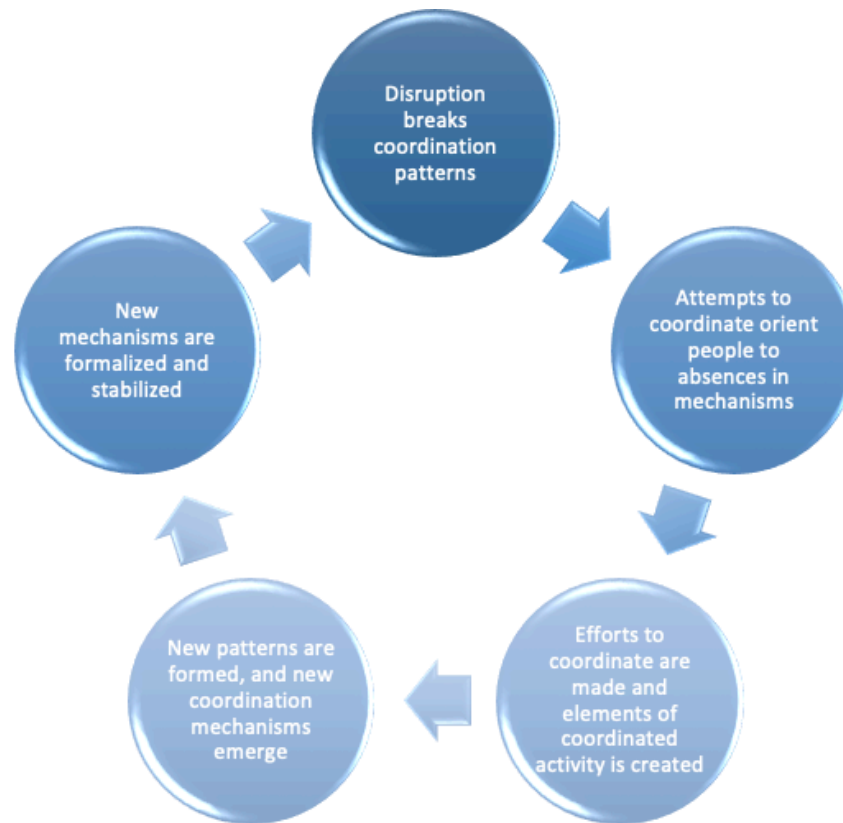


Figure 1. Creating coordination mechanisms through coordinating (adapted from Jarzabkowski et al. 2012).

2.2.5 Other theoretical approaches

The above sections have presented the theoretical approaches to coordination that have been most influential for the research in this thesis. However, because coordination of work has been studied for a long time across diverse research fields (cf. Okhuysen & Bechky, 2009; Wagner III, 2023; Zackrisson et al., 2015), there are several approaches that I have not been able to cover. In this section, I briefly describe a selection of other approaches to coordination that has been relevant in software engineering research. Notably, many of these approaches and traditions intertwine, and these brief descriptions are meant only as brief and admittedly superficial headings for sub-topics on coordination that were considered beyond the scope of this thesis summary.

Coordination as mutual adjustment. Thompson's (1967) seminal work on coordination in organizations represents an important theoretical underpinning in much modern research on coordination. In Thompson's conceptualization, coordination by mutual adjustment among interdependent actors is achieved using coordination mechanisms such as supervision, routines, scheduling, planning, and standardization. These mechanisms, however, have limited bandwidth and, accordingly, work best in settings where the level of uncertainty, as well as the level of task interdependence, is low (Gittell, 2006). Inter-team coordination, however, requires

a high bandwidth, which has led to the introduction of the term ‘layered mutual adjustment’ in research on large-scale agile (Dingsøyr, Rolland, et al., 2017; Moe et al., 2018).

Coordination modes. Another much-used approach within software engineering is Van de Ven and colleagues' three modes of coordination mechanisms (1976). In this influential sociological work, the authors investigate how task uncertainty, task interdependencies, and unit size (i.e., the coordinating unit, such as a team or an organization) influence coordination. Variations in these factors determine the use of coordination modes and mechanisms (Van de Ven et al., 1976). The coordination modes include the group mode, which refers to mechanisms based on mutual adjustment through feedback, for example, meetings (Dingsøyr et al., 2022); the personal mode, which refers to mutual adjustment based on feedback between two people, such as unscheduled meetings and ad hoc coordination (Moe et al., 2018), and the impersonal mode, which refers to the use of plans, schedules, and standardized information such as a project wiki (Strode et al., 2012).

Coordination and articulation work. Within the CSCW tradition, Schmidt and Simonee (1996) provided an influential view on coordination that focuses on the construction of coordination mechanisms in cooperative settings. Building on Malone and Crowston's work, they define coordination mechanisms as organizational constructs consisting of protocols, conventions, and procedures that are related to artifacts used to reduce and restrain the complexity of work in interdependent settings, what is referred to as ‘articulation work’ (Carstensen & Sørensen, 1996; Schmidt & Simonee, 1996). Within the software engineering field, this conceptualization has been used for investigating, for example, the role of communication software in global software engineering (Giuffrida & Dittrich, 2015) and in studies on continuous software engineering (Dittrich et al., 2018), all of which are relevant to large-scale agile.

Coordination through shared cognition. Typically researched from a teamwork perspective, shared cognition is a form of implicit coordination within groups of people that occurs without explicit communication (Rico et al., 2008). Research areas dealing with some form of implicit coordination by shared cognition span wide, including shared mental models (Schmidtke & Cummings, 2017), transactive memory systems (Lewis, 2003), expertise coordination (Faraj & Sproull, 2000; Faraj & Yan, 2006), and ‘the collective mind’ (Weick & Roberts, 1993). Within research on agile, Espinosa et al. (2007) studied team knowledge and coordination in distributed settings, and Strode et al. (2012) included implicit coordination in their theory of coordination in co-located agile teams.

Technical coordination in global software development. Global software development (GSD) and global software engineering have dealt with the topic of coordination in several ways using conceptualizations from other fields or sub-fields (e.g., Giuffrida & Dittrich, 2015; Li & Maedche, 2012; Stray & Moe, 2020). In addition to these, the work of Herbsleb and colleagues aimed at understanding coordination breakdowns in GSD led to the formulation of a socio-technical theory of coordination (Cataldo & Herbsleb, 2012; Herbsleb, 2016; Herbsleb

& Mockus, 2003) where coordination is understood as a problem of satisfying distributed constraints (i.e., dependencies) where people must “organize to solve this problem using capabilities and coordination mechanisms at their disposal” (Herbsleb, 2016, p. 6). This direction of coordination research aims specifically at understanding the technical dependencies and technical coordination in GSD (Cataldo & Herbsleb, 2012; Herbsleb, 2007; Kwan et al., 2011) but also relates to the shared cognition tradition (e.g., Espinosa et al., 2007).

A multiteam systems perspective on coordination. Finally, the multiteam systems (MTS) perspective (Marks et al., 2005) represents another approach to coordination that has been used in research on large-scale agile. An MTS is defined as two or more teams that interface directly and interdependently (Mathieu et al., 2002) and that heavily on cross-team coordination to accomplish collective goals (Marks et al., 2005). From this perspective, there are three basic coordination mechanisms, including mutual adjustment, direct supervision, and standardization (Wagner III, 2023), each of which is covered in previous sections. In research on large-scale agile, the MTS perspective has been used to study inter-team coordination (Scheerer et al., 2014) and the misalignment of dependencies (Bick et al., 2018).

2.2.6 The taxonomy of dependencies in agile software development

Regardless of the theoretical approach, coordination is conducted to deal with elements related to each other by some form of dependency. Some approaches, like Malone and Crowston’s CT (1994), explicitly include this relationship in the definition of coordination, whereas in other approaches, like RCT, interdependence is only implied. In a comprehensive review of coordination, Okhuysen and Bechky (2009) refer to coordination as “the process of interaction that integrates a collective set of interdependent tasks” (p. 463). The terms ‘dependencies’, and ‘interdependencies’ are used somewhat interchangeably across research fields. Within software engineering research, the term ‘dependencies’ appear to be the more common term, and this is

Table 6. A taxonomy of dependencies for agile software development (Strode, 2016) (From Paper 2).

Knowledge	A form of information is required for a project to progress	Requirement: Domain knowledge or a requirement is not known and must be located or identified.
		Expertise: Information about task is known only by certain persons or groups.
		Historical: Knowledge about past decisions is needed.
		Task Allocation: Who is doing what, and when, is unknown.
Process	A task must be completed before another task can process and this affects project progress	Activity: An activity is blocked until another activity is complete.
		Business process: Existing business processes cause a certain order of activities.
Resource	An object is required for a project to progress	Entity: A resource (person, place, or thing) is not available.
		Technical: A technical aspect of development affects progress, such as when two software components must interact.

also the term I tend to use in this thesis. However, the two terms should be understood as the same.

Coordination and dependencies are inseparable terms. However, this thesis focuses on coordination and coordination mechanisms as the means used to manage dependencies. I, therefore, limit the presentation of dependencies to the conceptualization used in this thesis. Other approaches focus on the dependencies as the primary unit of investigation.

In this thesis, I have adopted Strode's (2016) conceptualization of dependencies in agile. Following the work of Crowston and Osborne (2003), a dependency is defined as "a situation that occurs when the progress of one action relies upon the timely output of a previous action or the presence of a specific thing, where a thing can be an artifact, a person, or a piece of information" (Strode, 2016, p. 24). Based on her empirical work in agile teams, Strode developed a taxonomy of dependencies in agile, consisting of three main categories and eight sub-categories (Strode, 2016; Strode & Huff, 2012), presented in Table 6.

First, *knowledge dependencies* refer to when information is required for an individual or team to progress. This category is comprised of four sub-categories, that is, requirement, expertise, historical, and task allocation dependencies. Second, *process dependencies* refer to the order in which tasks or activities must be completed and consists of two sub-categories: activity and business process dependencies. Third, *resource dependencies* refer to the need for specific objects, including the two sub-categories of entity (a person, place, or thing) and technical dependencies (software and architectural components). These dependencies are managed using various coordination mechanisms. For instance, knowledge dependencies can be managed by stand-up meetings or product backlogs, process dependencies by burn-down charts and co-location, and resource dependencies (including technical dependencies) by "done" checklists and informal team communication (Stray, Moe, & Aasheim, 2019; Stray, Moe, Strode, et al., 2022; Strode, 2016; Vedal et al., 2021). This taxonomy is used in papers 2 and 3 and serves as the underlying dependency framework in Paper 4.

3 Research methods

This chapter describes the research methodology of this thesis. Section 3.1 presents the research perspective and approach, including a discussion of the choice of case study as the research methodology. Section 3.2 describes the case organization, and in Section 3.3, I present the data collection procedures. Section 3.4 is dedicated to data analysis. First, I provide information on Thematic Analysis, the overall analytical framework used in this thesis, before I present how the data was analyzed.

3.1 Research context

The research presented in this thesis is based on empirical data collected in a real-life large-scale agile software development setting. I have chosen to use the case study methodology, and specifically, I have conducted a single-case longitudinal case study relying on ethnographic data collection methods. This section presents the details and rationales behind these choices.

3.1.1. Research perspective

This thesis research is positioned within the research field of software engineering, which is a relatively young and multidisciplinary research field (Dittrich, 2016; Stol & Fitzgerald, 2015). As such, the theories and methods used in software engineering research often stem from other fields, such as the social sciences and the information systems field (Stol & Fitzgerald, 2015). In software engineering research, both quantitative and qualitative research methods are used to derive empirical knowledge, and case studies are common (Runeson & Höst, 2009).

Figure 2 presents an overview of the research paradigms, methodologies, and methods applicable to qualitative research. The figure is adapted from a course I took in qualitative research methods (Verne, 2021). To illustrate my position, I have circled the ones that apply to this thesis.

The upper lane in Figure 2 shows the three major research paradigms: positivist, interpretive, and critical (Klein & Myers, 1999; Runeson & Höst, 2009). This thesis research is conducted from an interpretive standpoint, meaning I attempt to understand phenomena by interpreting the research participants' (including my own) understanding of their context (Runeson & Höst, 2009). It is possible to conduct case studies from a positivist ontological and epistemological standpoint, where scientific knowledge is derived from 'objective facts' (Yin, 2018). However, the position that knowledge of reality and human action is a social construction resonates better with my understanding of the case, the data collected, and the analyses conducted for this thesis (Stake, 2005; Walsham, 2002).

I do not see the research falling under the critical paradigm, as I do not aim to provide any normative evaluations or directly examine any power structures (Klein & Myers, 1999). However, I could have performed critical research had I, for example, chosen to focus on the power structures shaping the coordination processes in the organization. Self-management and individual and team empowerment are defining characteristics of agile software development through the explicit focus on autonomous teams (Strode et al., 2012). In large-scale settings,

however, organizational structures are brought more to the front due to the multi-team and organizational-level characteristics of large-scale agile (Dingsøyr, Rolland, et al., 2017). Therefore, the role of managers and other organizational-level roles holding authority, such as software architects, are likely to be more salient in shaping what practices, activities, and mechanisms are considered important.

The main reason I consider the interpretive paradigm appropriate is the recognition that the data collected, the analyses, and subsequent presentations of the research findings, are not objective facts. Rather, they represent choices made based on my own understanding of the participants' descriptions of their realities and influenced by my participation in the field and engagement with the data material, as well as my evolving knowledge about the research topic (Crang & Cook, 2007; Walsham, 2006). The research could, and likely would, have unfolded differently if someone with another background than myself had conducted the study or if I were myself to conduct the study again. I will return to researcher positionality and my changing interpretations over time in Section 5.6.1.

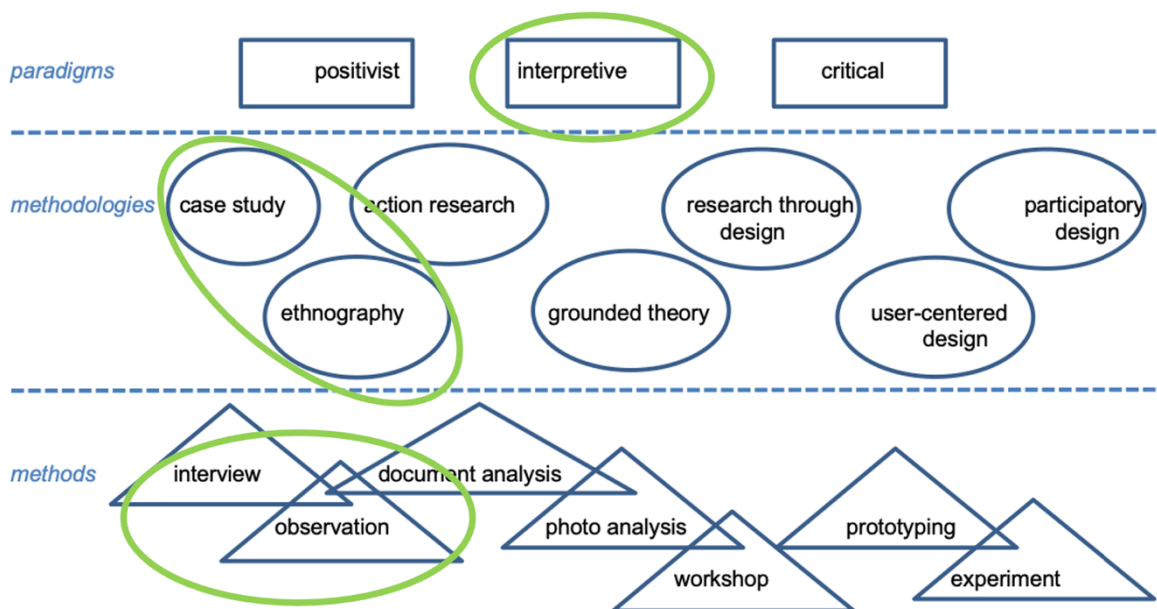


Figure 2. Overview of paradigms, methodologies, and methods in qualitative research (Verne, 2021).

3.1.2 Research approach

The middle lane of Figure 2 illustrates various research methodologies. This research was conducted as a single-case, qualitative, longitudinal case study that was strongly influenced by ethnography. Case studies are widely used in research, and numerous texts on the topic discuss everything from epistemological underpinnings to specific techniques (e.g., Easterbrook et al., 2008; Runeson & Höst, 2009; Yazan, 2015). In the following, I provide some definitions that have been relevant to my work, focusing on the qualitative and interpretive case study only.

According to Runeson and Höst (2009, p. 137), a case study has the following characteristics:

“1) it is of flexible type, coping with the complex and dynamic characteristics of real world phenomenon, like software engineering, 2) its conclusions are based on a clear chain of evidence, whether qualitative or quantitative, collected from multiple sources in a planned and consistent manner, and 3) it adds to existing knowledge by being based on previously established theory, if such exists, or by building theory.”

The research presented in this thesis demonstrates these characteristics. First, over the 1.5 years of fieldwork, many things happened within the case as a real-world phenomenon (i.e., the actual organization and the people working there). Therefore, it was essential to have the flexibility to change and adapt how I conducted the research to follow these changes. For example, having the flexibility to do field visits on other days than initially planned, reschedule interviews, observe new meetings that emerged, and so forth. Second, the conclusions of my research are all based on a detailed and longitudinal data collection from multiple sources and a long-term and detailed coding process. Details on the data collection and analysis are provided in the coming sections. Third, the case study adds to existing knowledge of coordination in large-scale agile as outlined in the contributions of this thesis.

According to Stake (2005), a case can be defined as a “bounded system” with certain features and activities that are inherently connected to the case. Outside features are considered significant as context but are clearly distinguished from the case. Following Stake’s (2005) definition, it is useful to view Entur, the case organization, as a bounded system in a complex internal environment. This understanding of the case allowed me to study the coordination practices and mechanisms used in the large-scale agile organization holistically, and to focus on what happened within the boundaries of the organization while at the same time seeing it as interrelated with its external context. I provide more details on the research case in Section 3.3.

The choice to conduct a single-case study.

Oftentimes, researchers are encouraged to conduct multiple case studies to increase case study validity (Yazan, 2015; Yin, 2018). This touches upon the tension and power relations between the positivist and interpretive research paradigms and quantitative and qualitative research methods (Flyvbjerg, 2006). Indeed, if the goal is to generate universally valid, objective facts, then knowledge derived from one single case will make for a poor research contribution. However, in line with scholars like Walsham and Geertz (Geertz, 1973; Walsham, 2002), I argue that ‘thick descriptions’ are equally, if not more valuable, when the goal is to understand *how* something happens, accepting that this ‘how’ is not *the only way* it could have happened.

When selecting a single case, then the case must be selected carefully. According to Stake (2005), cases are selected on the basis that the case is interesting prior to the formal study is conducted. They should be purposefully selected based on the researcher’s judgments of opportunities for intensive study and learning from the case. This was the case with Entur, as described in Sections 1.2 and 3.3. Flyvbjerg (2006) argues that single-case studies, if carefully selected and purposefully conducted, can contribute to scientific development by theoretical generalizability, provided that the case selected offers unique learning opportunities. He further outlines four types of cases, where the “critical case” provides opportunities for theoretical generalization. Flyvbjerg (2006) defines a critical case as a case that has “strategic importance

in relation to the general problem” (p. 229), which in this case is coordination in large-scale agile. I consider the case organization, Entur, a critical case because they are a large-scale agile development organization where multiple teams must coordinate to manage inter-team dependencies. This is a general characteristic that applies to other large-scale agile settings. Further, the complex internal and external organizational environment, to be described below, makes them likely to experience many of the coordination challenges outlined in the literature (e.g., Bick et al., 2018; Dikert et al., 2016; Dingsøy et al., 2018; Edison et al., 2022). Therefore, it can be argued that what is valid (or not) for this case would also be valid (or not) for many cases (Flyvbjerg, 2006).

An ethnographic approach to data collection.

While I first and foremost consider my research as case study research, I have taken an ethnographic approach to the data collection through the longitudinal fieldwork and the use of observation as a primary data collection method.

A defining characteristic of ethnography is researcher immersion in the context of their participants (Cragg & Cook, 2007; Sharp et al., 2016). In my data collection, I relied on methods central to ethnographies, such as long-term on-site presence, participative observation, extensive notetaking, semi-structured interviews, and more ad hoc-based field interviews. While my research could have been framed as ethnography per se due to these characteristics, I believe case study is more appropriate because my research questions and areas of interest are intrinsically related to the case as a bounded system (Stake, 2005), rather than the everyday practices and interactions of the people working in the case organization (Cragg & Cook, 2007; Sharp et al., 2016). Both aspects are certainly interesting and relevant; however, I see the former as more defining for the research project's focus. The research focuses on coordination and coordination mechanisms used in large-scale agile software development (i.e., the concepts), as represented through the case. An ethnography would traditionally focus more on the everyday actions performed and the meaning assigned to the activities by the participants (e.g., perceptions on the use of coordination mechanisms and their meaning from a participant's perspective). While some of my data certainly include such participant perspectives, it has not been my explicit focus. As a final note on this section, traditionally an anthropological method, ethnographies further involve cultural relativism (Cragg & Cook, 2007). While this aspect may appear less salient when conducting research in one's own national culture, I experienced some cultural aspects. Because my role as a researcher was very different from the developers, product owners, managers, designers, and testers' roles, I felt as an outsider that came to visit Entur to observe them. I think at least some of the employees there felt the same, at least in the beginning. As an outsider, it took time to familiarize myself with the language and terminology they used (they really did talk a lot about trains!). Additionally, I was introduced to an organization with its own culture that was different from the culture of my own organization at the University of Oslo.

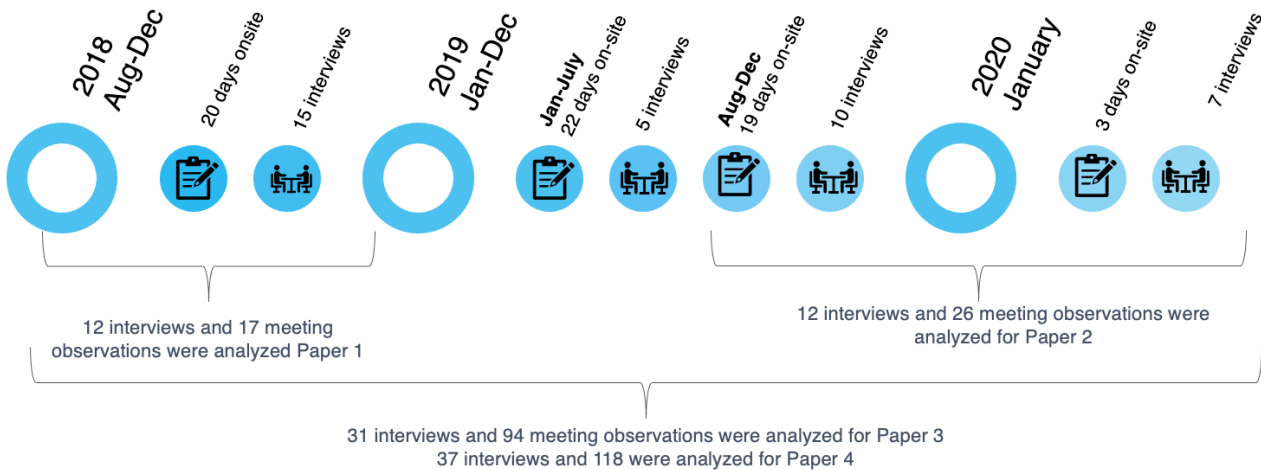


Figure 3. Data collection timeline

3.2 Data collection

I used three data collection methods: Observation, interviews, and collection of supplementary material. Together, these data sources have provided me with rich data that allows for interpretation from several angles (Crang & Cook, 2007; Stake, 2005).

3.2.1 Observations

Using observation as a data collection method allowed me to study coordination activities as they unfolded in the day-to-day environment in the case organization (Crang & Cook, 2007; Suri, 2010). My observation mode was open and partly participative in the sense that I was open to the participants about the general focus of my research. I observed both meetings and the day-to-day work at Entur. Over the 1.5 years, I spent 62 unique days on-site. During these days, I observed 108 meetings at all organizational levels. I tried to visit Entur regularly and spent, on average, one day per week with them. Figure 3 shows how the days were distributed.

While I participated in meetings and other on-site activities, and even in some off-site events, I did not perform any development work for the case company. The data from these observations are the detailed field notes I wrote during and after each day of fieldwork. I focused on getting as much detail as possible into the field notes, carrying with me a written notebook on all occasions. Over the span of the fieldwork, I filled almost ten A5-sized notebooks with handwritten field notes that I have kept for reference throughout the Ph.D. project. At the end of each fieldwork day (or as soon as possible after), I transferred the notes to my computer, both to have them digitally stored and to refine the notes to avoid losing out on details as memory fades and also to add any analytical thoughts that emerged during the rewriting process (Blomberg et al., 1993; Crang & Cook, 2007). I used an observation protocol (see Appendix A in Paper 4) to help structure the notes. These fieldnotes comprise a large data material corresponding to over 230 pages of text (standard margins, 11-point Calibri font). Appendix B provides details of the meeting observations.



Figure 4. A retrospective meeting at Entur.

3.2.2 Interviews

The data material includes 37 semi-structured interviews with representatives from all levels of the organization (Edwards & Holland, 2013; Walsham, 2002). I conducted 29 interviews, Dr. Stray conducted two interviews, and the six interviews with the agile method specialist were group interviews conducted by Dr. Moe, Dr. Stray, and me. The interviews were semi-structured, following an interview guide (see the appendices of papers 3 and 4) that was adapted depending on the interview participant and the focus of the interview. Fifteen interviews were held in 2018, 15 in 2019, and 7 in 2020. Twenty-six of the interviews were tape-recorded and later transcribed by me. Of the 11 that were not tape-recorded, six of them were group conversations among one or both of my supervisors and the agile methods specialist at Entur, and we did not consider the opportunity to tape-record them until later. The remaining five were ad hoc interviews where I did not have a tape recorder available. For all these interviews, the extensive note-taking technique described in the above section was used. The interviews lasted between 22 and 103 minutes, with an average of 52 minutes, corresponding to 31.5 hours of interviews in total. Appendix C provides details on the interviews.



Figure 5. A scheduled meeting at Entur.

3.2.3 Supplementary documentation

As an additional data source, I have collected various documentation used as supplementary material. For example, I collected company presentations, minutes from company meetings, examples of project documentation from Confluence and JIRA (software documentation tools), and various written information from the collaboration tool Slack (see Figure 19). I used this documentation to support factual claims made, for example, about the team organization (see Figures 8-10 for examples). I also took photos during the fieldwork that are used as illustrations (see Figures 4 and 5). I have not quantified this supplementary documentation because it is difficult to represent it systematically. Specifically, I had access to much of the written information in the case through my access to Slack, JIRA, and Confluence, but I did not systematically download all this information.

Table 7. Data material used for the different papers.

	Paper 1	Paper 2	Paper 3	Paper 4
Time period	August – December 2018	August 2019-January 2020	August 2018-January 2020	August 2018-January 2020
Observations	17 meetings and fieldnotes from 20 days on-site	26 meetings and fieldnotes from 22 days on-site	94 meetings	108 meetings and fieldnotes from 62 days on-site
Interviews	12 interviews	12 interviews	31 interviews	37 interviews
Supplemental material	Slack logs	Product backlogs, prioritization documents, Slack logs, meeting agendas	Slack logs, program documentation, e-mails, internal and external company documents (e.g., presentations, reports)	

Detailed analyses were carried out for each research paper included in this thesis. Table 7 shows how portions of the data material were used, and new analyses were conducted for each paper. Paper 1 was written while the fieldwork was still in progress, whereas papers 2 – 4 were written after the data collection phase had ended. Figure 10 illustrates how codes were generated during the phases of the thematic analysis across the thesis research.

3.3 The research case

In this section, I present details about the case organization. This section is meant to give a general overview; more details are found in the research papers.

3.3.1 The case organization

The case study was conducted in Entur, a Norwegian public sector organization. Entur was established in 2016 following a political reform initiated by the Norwegian Ministry of Public Transportation. The political reform entailed splitting up the Norwegian railroad network operation, which had been run in its entirety by one company until now. With the reform, public tenders were announced for the different railroads (for example, the route between Oslo and Trondheim), and both national and international train operators were allowed to participate in the tenders. In response to this, Entur was established to gather all of Norway's public transport services, offering services to the public transport operators and the general public (see entur.org for more information).

When I started the Ph.D. project in 2018, Entur had already been in touch with SINTEF in relation to the Norwegian Research Council project A-team. When we started discussing possible options for data collection for my research project, Entur quickly became an interesting candidate for conducting fieldwork on coordination in large-scale agile. In August 2018, Dr. Stray and I met with Entur representatives to set up arrangements. During these initial meetings, we learned more about the organization, the team organization, and their challenging areas. Following these meetings, my fieldwork started in late August 2018. I followed Entur for 1.5 years until the end of January 2020.

In the following, I describe some reasons why Entur caught our attention as a relevant case for studying coordination in large-scale agile. First, they were (and are) organized as an ongoing large-scale development program with a complex external environment, a complex product, and many dependencies across teams, making it an interesting case for studying inter-team coordination. Moreover, we found Entur interesting because they were registered as a software company working with public transportation (rather than a public transportation company with an IT department). Further, they started out working with agile methods right from the beginning rather than undergoing an agile transformation. From a coordination perspective, Entur was interesting because of the dependencies associated with their software development. The teams needed to work together to deliver their products, and many teams depended on each other to progress on their parts of the system. Entur also depended on one of their clients because the old software system was closely tied to this company. Further, being a public sector

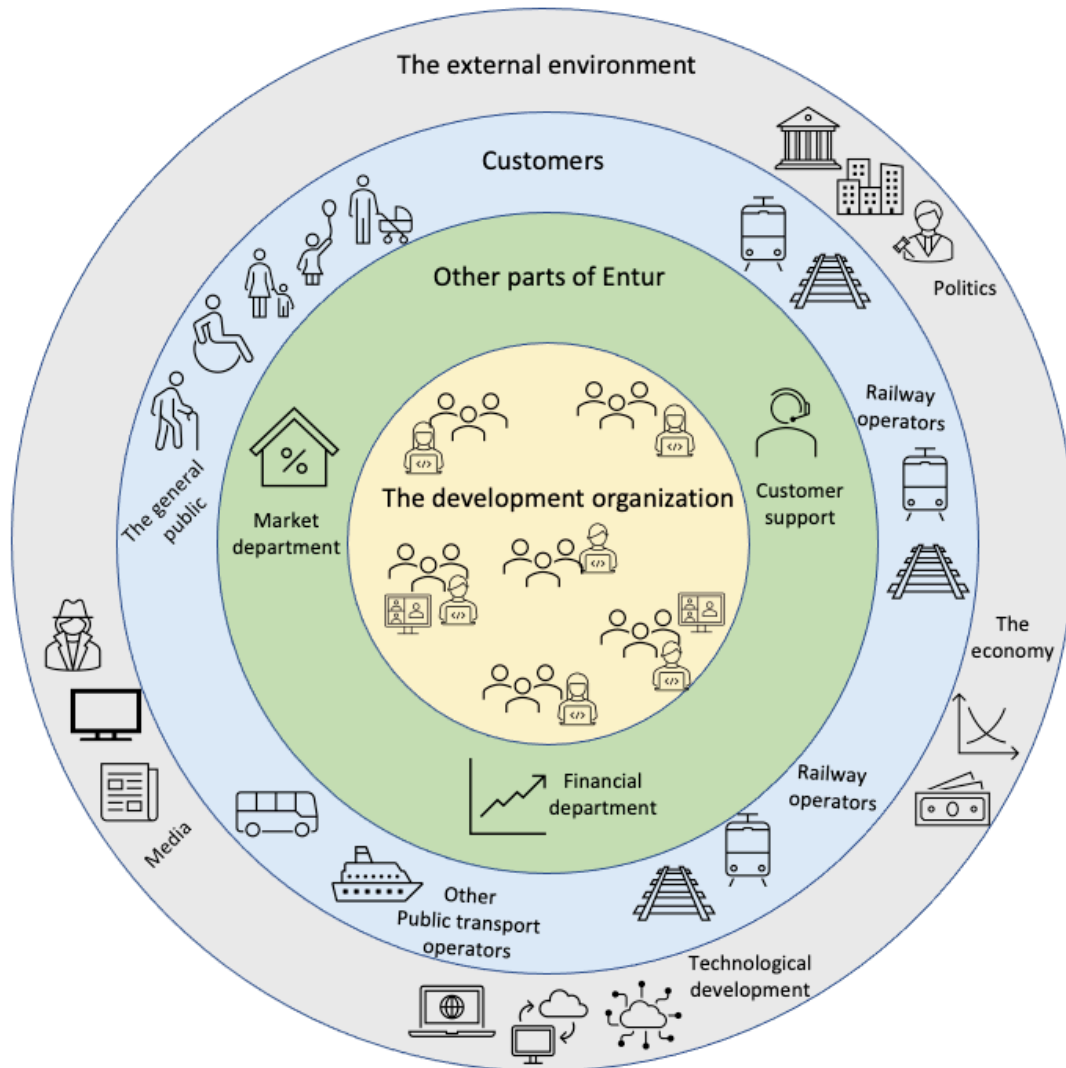


Figure 6. Enturs internal and external environment.

organization with political underpinnings, Entur's financial situation was also complex and partly dependent on the political situation in Norway. The political backdrop also shaped Entur's software deliveries to a certain extent, in that some hard deadlines were set by external parties. Furthermore, the media would sometimes have an interest in reporting on the progress of the reform and Entur's deliveries. All in all, Entur had many features of a critical case for studying coordination in large-scale agile development (Flyvbjerg, 2006). Figure 6 illustrates Entur's internal and external environment.

3.3.2 The software products

Entur develops several software products, including a multi-platform travel planner where users can manage their entire trip within a single application (See Figure 7). Another product is a new sales platform and API that railway operators and other public transportation operators can use to distribute their products to the public through the Entur app or their own channels. They also manage the physical systems for selling and distributing railway tickets (i.e., software and

firmware for ticket booths and hand-held ticket machines used on the trains). Finally, they collect, refine, and share public transportation data through open APIs².

As such, Entur customers and users include both public transportation operators in Norway that use their APIs and sales systems, as well as the general public. Their products are under ongoing development and updated, and new functionality is continuously released.

While the new sales platform was under development, an old system inherited from the company that previously ran all railroads was kept running. This was done in order for Entur to provide their services. The new platform was cloud-based and built on modern architectural principles. It was based on microservices, running on the Google Cloud Platform with Kubernetes and Firebase. This old system, which was based on a monolithic architecture, required ongoing maintenance, and features developed for the new platform also needed to be compatible with the old system. In other words, there were many dependencies between the systems while Entur was working towards making the old system redundant.

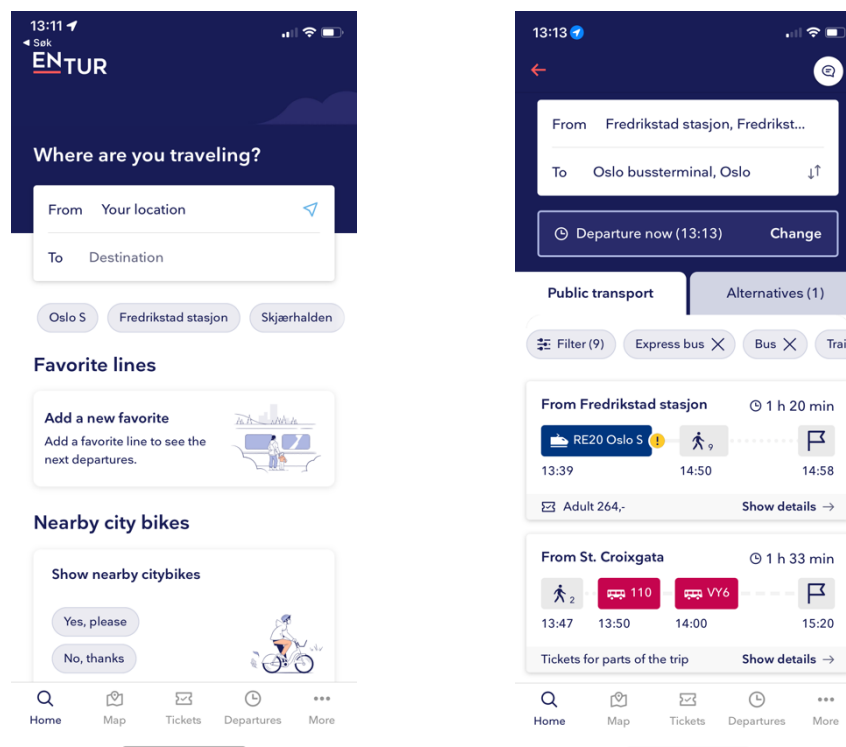


Figure 7. Screenshots from the Entur mobile application (14.02.23).

² See www.entur.org for more information.



Figure 8. The early team organization (approx. 2016-2017).

Entur used many programming languages and technical tools to develop the new platform. They adopted new technologies as needed, and during the fieldwork period from 2018-2020, central languages included Kotlin, Java, and Scala for back-end, and JavaScript (Node.js), and React-Native for front-end. Additionally, they used support tools such as Grafana, Prometheus, JIRA, Confluence, and Slack.

3.3.3 The team organization and the large-scale agile environment

Entur started working with agile methods from day one. Despite being a large-scale agile organization from the outset, Entur has chosen not to adopt any scaling framework (such as SAFe or LeSS). Instead, they tried to tailor to their specific needs, using best practices from DevOps and continuous software engineering, such as continuous deployment and integration, and they gained inspiration from companies such as Spotify and Google. Practices were subject to change as the organization scaled and new needs arose. From August 2018 to January 2020, the number of development teams grew from 13 to 17, and the number continued to grow after 2020.

Entur organized the developers into teams that each had areas of responsibility towards the overall product. The development teams were cross-functional but focused on different parts of the overall deliveries, such as sales, ticketing, and pricing. In addition to the development teams, there was a separate test team, and temporary task force teams were established when needed. Team size and composition varied slightly over time; the largest team had more than 16 members, while the smallest had about five.

Case representatives reported that, on average, the teams spent 40% of their time developing new features and 60% on maintenance, bug fixing, and improving the code (i.e., reducing technical debt). Each development team had a team leader, product owner, tech lead, and developers. Some team leaders and product owners were responsible for more than one team, as shown in Figure 9. The number of members per team ranged from five to 17. In addition to the team roles, there were roles at the inter-team level, such as program managers, architects, and customer managers. The teams worked in an open office landscape that was also used for open space sessions as well as for displaying inter-team tools and artifacts.

Overall, the teams had the autonomy to choose how to organize themselves and which agile practices, tools, and techniques to use in solving their team-specific development goals. Practices from Scrum and Kanban, such as stand-ups, retrospectives, product backlogs, and visual task boards, were commonly used. An important factor for the use of agile methods in

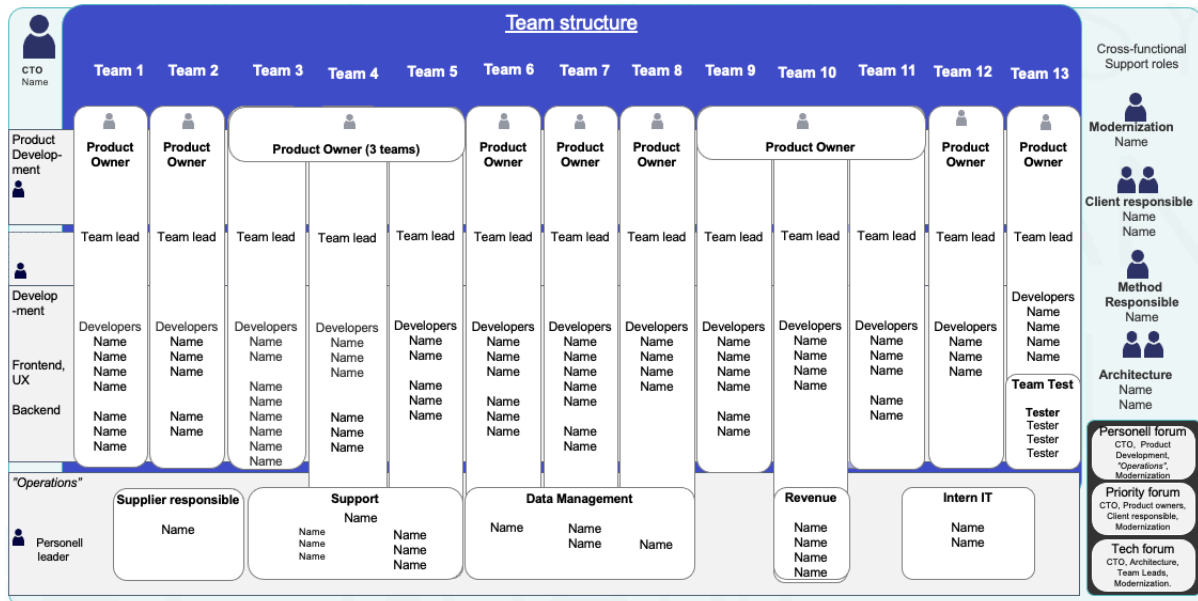


Figure 9. Enturs matrix team organization 2018-2020, adapted from a company presentation.

the program was the support of top management and the board of directors to work in this way. Another was their ability to test and experiment with their ways of working to respond to their internal and external environment while simultaneously keeping up to speed in delivering services to their clients and the public. As such, the approaches to software engineering and agile practices in the program were not static but changed over time. This meant some practices emerged as the program scaled, whereas others disappeared.

Throughout the data collection period, Entur underwent several changes. In addition to the scaling of the program in terms of teams, the organization went through re-arrangements of the organization structure, from a sequential team organization in 2016, with agile practices within the teams but sequential handovers between teams (Figure 8), to a matrix organization in 2017 (Figure 9), and towards organizing into product areas from 2020 onwards (Figure 10). In 2019, they moved offices to fit the growing number of teams. More details on the changes in the team organization are presented in Paper 4.

In sum, Entur’s complex and ever-changing large-scale environment, filled with a range of various dependencies within and across teams, made it a unique and interesting case for studying coordination and coordination mechanisms in large-scale agile.

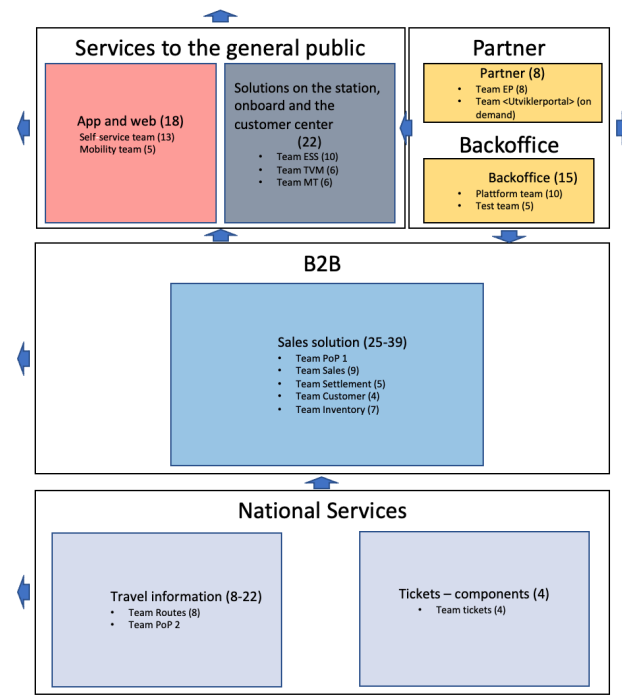


Figure 10. Enturs team organization in service areas from around 2020, adapted from a company presentation.

3.4 Data Analysis

In this section, I explain the data analysis in this thesis at an overall level. Details on the separate analyses are introduced in each of the four research papers.

3.4.1 Thematic analysis

The overall analytical approach taken in this dissertation is thematic analysis. This version of thematic analysis was popularized by a 2006 paper written by psychology researchers Virginia Braun and Victoria Clarke (Braun & Clarke, 2006). This paper became “unexpectedly popular”, and during the past two decades, the method has been widely used beyond the field of psychology (Braun et al., 2022), including in the software engineering field (e.g., Ågren et al., 2022; Hussain, Perera, et al., 2022; Munir et al., 2016; Stray, Florea, & Paruch, 2022; Wohlin & Aurum, 2015). In later years, Braun and Clarke coined the term “reflexive thematic analysis” to clarify how this form of thematic analysis differs from other forms (Braun et al., 2022; Braun & Clarke, 2019, 2021b). In this thesis, I use the term thematic analysis for simplicity.

I chose thematic analysis because a versatile and flexible analytical framework suitable for case study research with large amounts of data. I also chose it because it fits well with the interpretive underpinnings of the research. Thematic analysis is a method for systematically working with large and varied research data (Braun & Clarke, 2012). The method provides tools for identifying and analyzing patterns and commonalities across the data corpus to form meaningful themes that can be used to understand qualitative research questions. A *theme* can

be defined as a central organizing concept that can be used to capture recurring patterns of a similar type within the data. This can be characteristics, properties, modes of usage, or any recurring pattern around which data items and codes can be organized around (Braun et al., 2022; Braun & Clarke, 2012). For example, various tools that are used to coordinate can become the theme “coordination tools.”

Both inductive (i.e., deriving themes directly from the data) and deductive (i.e., building on existing theories and knowledge) approaches are compatible with thematic analysis. Within a larger thematic analysis such as this dissertation, a combination of the approaches is encouraged, thereby providing strong links to the empirical data and to existing knowledge on a particular research theme (Braun & Clarke, 2006, 2012). I followed this recommendation, relying on the knowledge derived from the literature on coordination and large-scale agile while at the same time remaining open to the interpretations that could be made from the data.

For all papers, Malone and Crowston’s (1994) definition of coordination as the management of interdependent activities has been used as the underlying definition. Further, in Papers 2, 3, and 4, I have relied on the taxonomy of dependencies for agile teams developed by Strode (2016), i.e., using the terms *knowledge*, *resource*, and *process* dependencies and their sub-categories (as introduced in Chapter 2). In addition, I have applied existing theoretical frameworks when analyzing the research data. For Paper 1, I used Relational Coordination Theory (Gittell, 2006) and analyzed the data in light of the concepts provided by this theory. For Paper 2, I used the theory of coordination for co-located agile projects (Strode, 2016; Strode et al., 2012), adapted to the inter-team level.

In Paper 3, a more inductive approach is taken. Ideas of software development as a socio-technical activity (Herbsleb, 2007; Hoda, 2021; Storey et al., 2020) inspired the data analysis. However, it was the data material that drove the development of the taxonomy of inter-team coordination mechanisms and the TOPS framework.

In Paper 4, I used the theoretical framework of Jarzabkowski et al. (2012) as an analytical backdrop. However, I also worked with the data inductively to extend the findings, as the Jarzabkowski framework did not capture all the themes we found during the analyses. In this paper, I also used the coordination mechanism themes (i.e., meetings, roles, and tools and artifacts) derived for Paper 3 and analyzed them again, focusing on change. The resulting model for analyzing changes in coordination mechanisms in large-scale agile is therefore based partly on existing theoretical knowledge and partly on the empirical data.

Importantly, in thematic analysis, themes are not thought to “emerge” from the data. Instead, the active role of the researcher in searching for and constructing themes from the data is explicitly recognized (Braun et al., 2022; Braun & Clarke, 2012). On a related note, the term ‘data saturation,’ which is often associated with qualitative analyses, is typically not used within the thematic analytical framework (Braun & Clarke, 2021b). This is because of the explicitly recognized constructivist role of the researcher: It is the researcher’s responsibility to decide when to stop analyzing, not based on a defined state of “no new information is emerging” (as is common, for example, within grounded theory terminology (Hoda, 2021)), but rather on a qualified judgment call when the themes are complex and rich enough to justify the definitions of the themes derived during the later phases of the analysis (Braun & Clarke, 2006, 2021b).

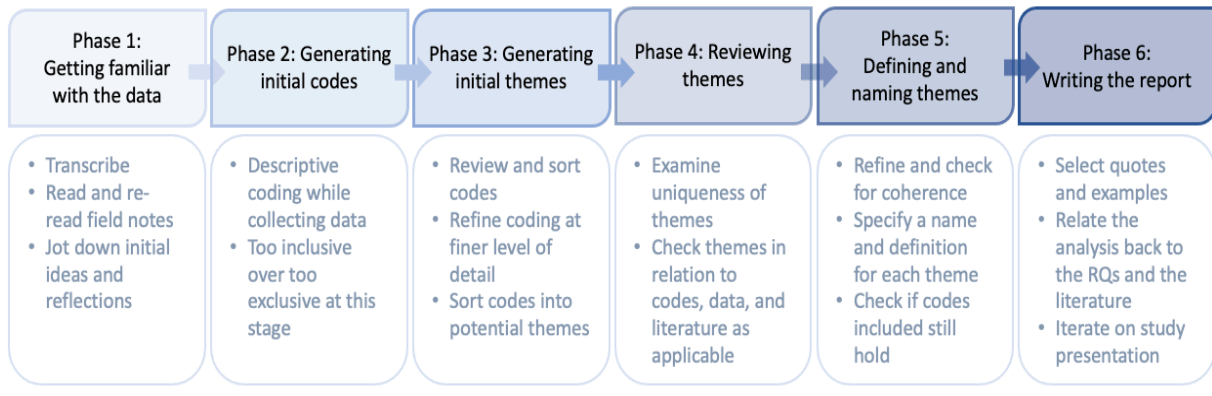


Figure 11. Overview of the six phases of thematic analysis.

In the thematic analytical framework outlined by Braun and Clarke (2006, 2012), the analysis is carried out across six distinct but interrelated phases. Each phase brings the researcher closer to a fuller understanding of the themes constructed from the data. The following paragraphs and Figures 11 and 12 provide details.

Phases 1 and 2: Getting familiar with the data and generating initial codes.

The thematic analysis starts with getting familiar with the data material. In line with the ethnographic approach, data analysis began at one level immediately after the observation started and was conducted continuously throughout the data collection (Crang & Cook, 2007). As such, most initial coding was performed while the fieldwork was still in progress. I transcribed, read, and reread the material and noted initial ideas regularly throughout the data collection period. These initial analyses were conducted in the form of writing analytical notes (often referred to as analytical memos, for instance, within the Grounded Theory methodology) directly after field observations and interviews to capture any relevant piece of information, as well as my own interpretations, analytical thoughts, and ideas (Crang & Cook, 2007). This familiarized me with the data and allowed for initial analytical reflections on how coordination was performed while the fieldwork was still in progress. I regularly discussed my thoughts and ideas with my supervisors to help structure this early analysis.

The next phase, generating initial codes, was partly performed alongside writing analytical notes. This form of coding was very descriptive and high-level. A key point at this stage is to be open and inclusive as to what meaning the data material holds. Therefore, I did not limit myself to any number or types of codes, considering that it was better to be too inclusive over too exclusive, as codes were refined in later phases. I also repeated this phase for each of the four papers, where I started to focus the coding closer to the specific research questions.

My supervisors were involved in this phase, supporting my initial coding through regular discussions. Papers 1 where my supervisors were the only co-authors, was written while the fieldwork was still in progress. The early coding for Paper 1 was done partly during a co-located writing workshop, where we went through and coded the first couple of the twelve interview together, before I took over the remaining coding of the data. They reviewed and helped me refine the coding in line with the research questions. As my experience with data coding and data analysis grew with each research paper, my level of independence expanded. For the three

remaining papers, I performed the initial coding and discussed questions and issues raised from the process with supervisors and other co-authors. For the final analyses presented in this thesis summary, I coded the material independently.

Phases 3 and 4: Generating initial themes and reviewing themes.

During these phases, the codes generated during the second phase and the researcher's detailed knowledge of the data are used to group codes into initial themes that say something meaningful about the data (Braun & Clarke, 2006, 2021a). This was done by reviewing, refining, and regrouping codes to identify themes for each part of the thesis, either in relation to an underlying theoretical lens or more inductively based on the data itself. For example, in Paper 1, the concepts from RCT, whereas in Paper 4, themes were related to codes related to change as identified from the data. An example of the deductive approach is using the theoretical concept of "shared knowledge" from Relational Coordination Theory, coding all instances where "shared knowledge" applies, and grouping these under one theme of "shared knowledge." An example of how I used the inductive approach was to group all different types of meetings where coordination was relevant into the larger category, or theme, "coordination meetings".

In moving from generating to reviewing, themes were checked in relation to the coded extracts, the data, and the literature, depending on the focus of each paper. During this phase, the uniqueness of the themes and the codes included were evaluated, and similar and overlapping themes were identified. For example, in the analysis of the whole data material conducted for Paper 3, I initially identified 59 potential coordination mechanisms used within and across teams. During the phases of the thematic analysis, this was reduced to the 27 inter-team coordination mechanisms reported in Paper 3.

The supervisors, as well as other co-authors, were involved, depending on the paper. In Paper 1, my supervisors worked tightly with me ensuring a good anchoring between the RCT elements and the data material. For Paper 2, I was in full charge of the analytical process, but we held regular discussion meetings where we reviewed and discussed themes. For Paper 3, similar discussion meetings were held with the second author, Dr. Hoda, while the two supervisors were not involved until the later stages of the process. In Paper 4, these two phases were largely driven and conducted by me, involving the co-authors mostly for brainstorming and quality assurance. For the analysis conducted for this thesis summary, I went back to the original list with 59 mechanisms to analyze not only the inter-team mechanisms, but all mechanisms used for coordination in the large-scale agile development program. I critically examined the uniqueness of each potential mechanism, combined duplicates, and removed what did not fit the definition of coordination mechanism and did not manage any dependencies. This new analysis resulted in a set of 47 mechanisms, as reported in Chapter 4.1.

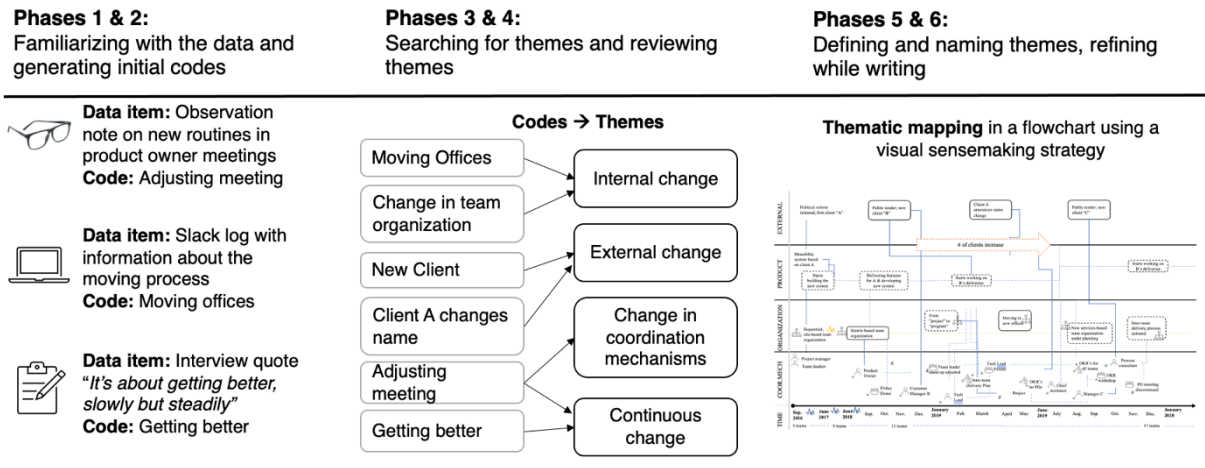


Figure 12. Illustration of the thematic analysis for Paper 4.

Phases 5 and 6: Defining and naming themes and writing the report.

When moving to the final two phases, the analysis starts to “set” in that the themes are named and defined, setting the boundaries for what can and cannot be included in the theme. Ideally, all codes and higher-level themes should fit in a coherent thematic map at this stage (Braun & Clarke, 2006). However, should anything not make sense, it is possible to move back to a previous analytical stage and conduct further analyses until the composition of themes again makes sense (Braun et al., 2022; Braun & Clarke, 2012). For each paper, the specifics of each theme were refined and checked for coherence. My co-authors were involved in a manner similar to what is described for phases 3 and 4.

In the final phase, the analytical findings are further refined as the report is written. Here, the writing up of the results and discussion section often provides a final “sanity check,” also as co-authors often provide new input on less clear passages or things that may not make sense in writing (Braun & Clarke, 2012). Moreover, during the write-up, the most compelling quotes and examples are selected to illustrate the findings, which provide further refinement. Due to the iterative nature of thematic analysis, it is still possible to go back and re-analyze parts of the data should a theme not make sense in relation to the other themes during the write-up, as thematic analysis provides analytical opportunities right until the report is finished (Braun & Clarke, 2006, 2021b). This is demonstrated by the new insights I gained from the final analysis reported in Chapter 4.

4 Findings

This chapter presents the answers to the three research questions of this thesis. Section 4.1 provides an overview of which coordination mechanisms are used to manage inter-team dependencies at Entur (RQ1). Section 4.2 goes into detail on the how dependencies are managed as related to their technical, organizational, physical, and social (TOPS) characteristics. and provide a detailed analysis of three example mechanisms (RQ2). Finally, in response to RQ 3 Section 4.3 presents a framework for analyzing coordination mechanisms in large-scale agile.

4.1 RQ1: Which coordination mechanisms are used to manage inter-team dependencies in large-scale agile software development?

To answer RQ1, I went back to the analyses conducted for the four papers and re-analyzed all previously identified coordination mechanisms. For example, during the analysis that led to the taxonomy of inter-team coordination mechanisms (Figure 14), I identified 59 potential mechanisms used at Entur, which were reduced to the 27 inter-team coordination mechanisms reported in Paper 3. For this new analysis, I went back to these mechanisms, as well as the mechanisms reported in the other papers, and re-examined each of them. During this new analysis, I identified 47 coordination mechanisms used to manage inter-team dependencies, of which ten are not previously reported in the papers.

As shown in Figure 13, there were twenty-one meetings, thirteen roles, and thirteen tools and artifacts. Seven mechanisms were located at the within-team level, 34 at the inter-team

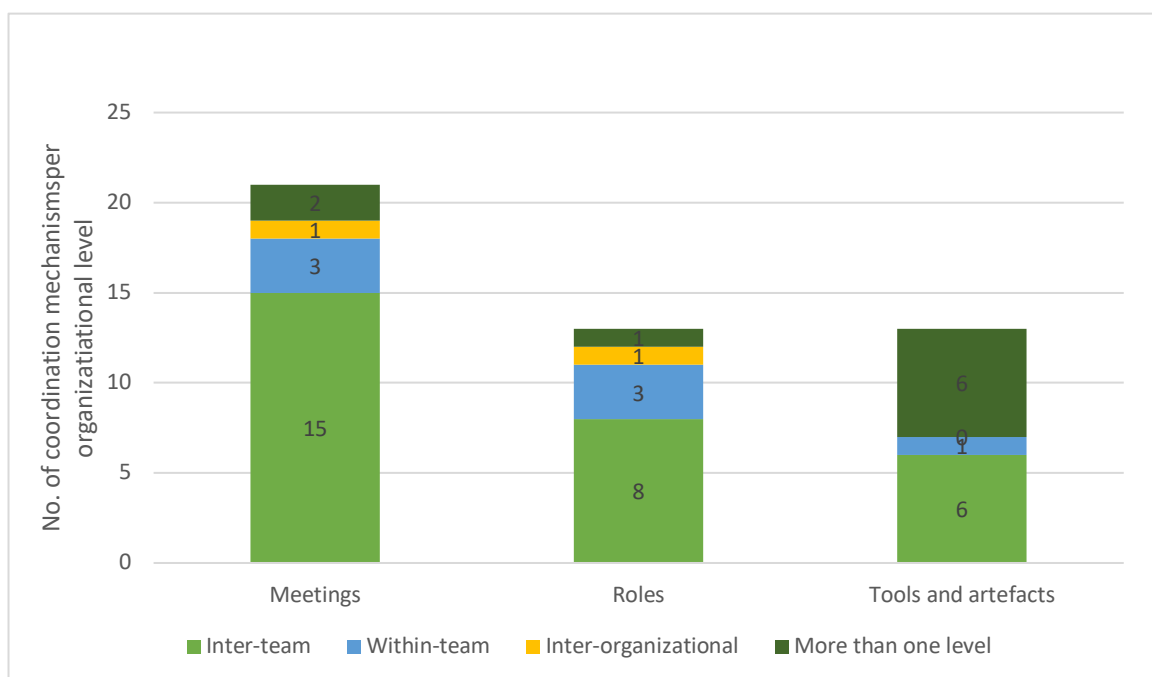


Figure 13. Distribution of organizational level of the coordination mechanisms.

level, and two at the inter-organizational level. Twelve mechanisms operated at more than one level. The mechanisms are categorized according to the taxonomy of inter-team coordination mechanisms, including coordination meetings, roles, and tools and artifacts. Tables 8-10 provides overview of each of the 47 coordination mechanisms listed according to these categories. Each table provide a short description of the coordination mechanisms, a categorization of organizational level, a characterization of their TOPS characteristics, and which types of dependencies they manage. The dependency analysis is based on the work of Strode (2016), as presented in Section 2.2.6, using the categories of knowledge, process, and resource dependencies.

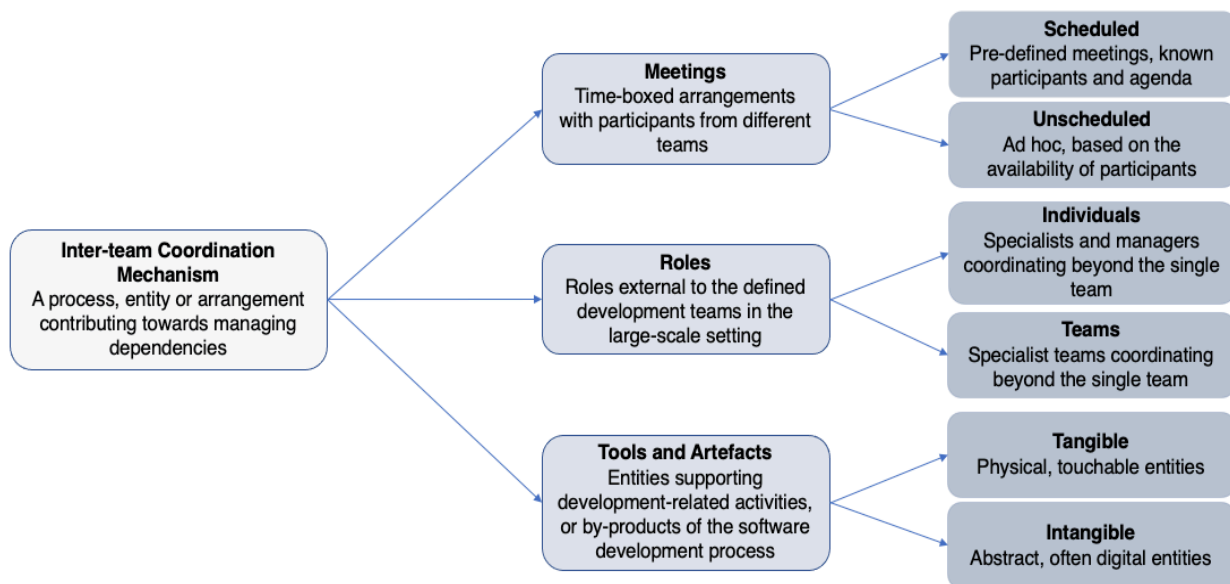


Figure 14. The taxonomy of inter-team coordination mechanisms in large-scale agile (From Paper 3).

4.1.1 Coordination meetings

The first category, coordination meetings, are defined as time-boxed or ad hoc arrangements where inter-team dependencies are managed by enabling people to discuss, share knowledge and negotiate shared understandings. From the case data, I identified 21 unique coordination meetings that served to manage different inter-team dependencies. Table 8 presents the coordination meetings in more detail.

Thirteen of the meetings were previously reported in one or more of the four research papers, whereas eight were identified during the new analysis. This relatively high number of new elements included are based on my evolving understanding of what constitutes an inter-team coordination mechanism, as will be detailed in Section 4.1.4 below.

Seventeen of the meetings were classified at the inter-team level, meaning that participants from at least two different development teams were present. Additionally, four types of meetings were conducted at the within-team level, and three at the inter-organizational. Three types of meetings operated at more than one level, meaning that different variants of the meeting were conducted at each organizational level. For example, ad hoc, unscheduled

coordination meetings (Table 8, mechanism no. 9), where the topic of discussion was inter-team dependencies, happened at all organizational levels.

The coordination meetings serve to manage various aspects of knowledge, process, and resource dependencies (Strode, 2016). All the 21 meetings contributed to managing knowledge dependencies. This is perhaps not surprising, as meetings per definition allow people to get together to share information and reach a shared understanding of how to proceed to solve between-team issues in the development process. Another reason is the high-level analysis of the dependencies. The knowledge dependency category in Strode's (2016) dependency taxonomy contains four different sub-categories of knowledge dependencies (i.e., requirement, expertise, historical, and task allocation dependencies, see Section 2.2.6). A more fine-grained reporting would have led to more nuances. For example, the product owner prioritization meeting (mechanism no. 3 in Table 8), primarily relates to the task allocation knowledge dependency, as the goal of the meeting is to coordinate task prioritizations across teams. As another example, mechanism no. 10 in Table 8, the client status meeting, can be related to the expertise knowledge dependency as the presence of certain expertise (i.e., the clients) are required. This meeting is also related to the requirement knowledge dependency, as information from clients about product requirements is a central part of the meeting. This type of classification could have been detailed for all mechanisms. However, because the focus of this analysis is to present the novel findings of my own dissertation research (rather than demonstrating the applicability of the dependency taxonomy by Strode), I decided to keep the reporting at a general level.

Similarly, eighteen of the 21 meetings were assigned to managing resource dependencies. Resource dependencies consist of entity and technical dependencies (Strode, 2016). As the goal of most meetings were to resolve technical or software product-related issues between the development teams, most meetings related to resource dependency management. Finally, process dependencies are made up by activity and business process dependencies (Strode, 2016). A total of eight of the 21 meetings managed either of these dependencies at an inter-team level. This low number compared to the other two dependency categories is explained by the more project management and business-related aspect of these types of dependencies, which were not natural to discuss during many of the meetings which were more oriented towards software development.

Table 8. List of all coordination meetings observed during the fieldwork.

Coordination Meetings							
Time-boxed or ad hoc arrangements where inter-team dependencies are managed by enabling people to discuss, share knowledge and negotiate shared understandings.							
Coordination mechanism name	Organizational level	T	O	P	S	Description	Dependency Managed
1 Friday Demo ^{2,3,4}	Inter-team	✓	✓	✓	☑	A weekly demo for all employees. Teams take turn showcasing their work, demonstrating new features across all teams. An informal arena for socializing, often with snacks provided. Conducted in a large open space with audio-visual arrangements.	Knowledge
2 Product Owner weekly meeting ^{1,2,3}	Inter-team	✓	✓	✓	☑	Product owners meet bi-weekly during lunch hours in a meeting room close to the cantina. Discussion of technical product, as well as organizational topics, managing resource, process, and knowledge dependencies.	Knowledge, process, resource
3 Product Owner Prioritization meeting ^{1,2,3,4}	Inter-team	☑	✓	✓		Bi-weekly, conducted in front of a prioritization task board. Focused on product and technical requirements. In late 2019, the meeting was replaced by a stand-up.	Knowledge, resource
4 Product Owner workshop ^{1,3}	Inter-team	☑	✓	✓	✓	POs meet quarterly to plan and discuss longer-term technical product-related areas. Organizational issues, such as team structure, are also discussed. Held at an off-site location and includes retrospectives and informal socializing.	Knowledge, process, resource
5 Team-level status meetings ¹	Within-team	☑	✓	✓		Weekly status meeting with a focus similar to daily stand-ups, but with a wider focus. Primarily related to product progress, dependencies to other teams were regularly discussed. Mostly co-located, but adapted based on team member location.	Knowledge process,
6 Team-level retrospectives ¹	Within-team		✓	☑		Held approximately monthly, with a primary focus on teamwork, cooperation, and coordination. Primary focus on within-team matters, but inter-team dependencies were discussed as relevant. Mostly co-located, but adapted based on team member location.	Knowledge, process, resource
7 OKR workshops ^{3,4}	Inter-team	☑	✓	✓	✓	Held quarterly at an off-site location to discuss, align, set, and share inter- and intra-team OKRs. OKRs primarily relate to technical (product) progress, but can also be related to organizational outcomes,	Knowledge, process
8 Program architect meeting ³	Inter-team	☑	✓	✓	✓	Weekly meeting where product technical and architectural quality are recurring themes. Organizational aspects can also be discussed, thereby managing primarily resource, but also process dependencies.	Knowledge, resource, process
9 Unscheduled conversations/ meetings ^{1,3}	Within-team, inter-team, inter-organizational	✓	✓	✓	☑	Conducted ad hoc, as needed. Typically focus on product features and deliveries, or organizational aspects, thus managing knowledge, process, and resource dependencies. Conducted in open office space or meeting rooms.	Knowledge, resource

10	Client status meetings ^{New}	Inter-organization al	<input checked="" type="checkbox"/>	✓	✓	Held weekly and primarily concerned with technical product progress. Held at client site.	Knowledge, resource
11	Team-level stand-up meetings ^{New}	Within-team	<input checked="" type="checkbox"/>	✓	✓	Held approximately every day, and mostly concerned with team-level product-related (technical) issues. However, relevant dependencies to other teams were addressed as needed. Mostly co-located, but adapted based on team member location.	Knowledge, resource
12	Inter-team/Task force sprint planning ^{New}	Inter-team	<input checked="" type="checkbox"/>	✓	✓	When task force teams were used, these held their own sprint planning meetings. During these meetings, members of different teams would sit together (physically or digitally). Focus on product, thereby managing technical resource dependencies.	Knowledge, resource
13	Client retrospectives ^{New}	Inter-team, inter-organization al	<input checked="" type="checkbox"/>	✓	✓	During these retrospectives, Entur representatives from different teams and inter-team roles, and client representatives gather to discuss the product, process, and progress with focus on the client and deliveries, thereby managing knowledge, resource, and process dependencies.	Knowledge, resource, process
<p>Inter-team retrospectives³: In Paper 3, this referred to any kind of inter-team retrospectives. However, because different inter-team retrospectives may serve different purposes and manage different dependencies, they are now expanded as detailed below.</p>							
14	Product owner retrospectives ^{New}	Inter-team	<input checked="" type="checkbox"/>	✓	✓	At these retrospectives, which were preferably co-located, the product was in focus, but also organizational and inter-personal matters could be discussed, thereby managing all here types of dependencies.	Knowledge, resource, process
15	Team leader retrospectives ^{New}	Inter-team	<input checked="" type="checkbox"/>	✓	✓	At these retrospectives, which were preferably co-located, the team leader collaboration was in focus, but product-related matters could be discussed, thereby managing technical resource and knowledge dependencies.	Knowledge, resource
16	Tech lead retrospectives ^{New}	Inter-team	<input checked="" type="checkbox"/>	✓	✓	At these retrospectives, which were preferably co-located, the tech lead forum was in focus, thereby managing technical resource and knowledge dependencies.	Knowledge, resource
<p>Community of practice meetings³: In Paper 3, this referred to all communities of practice. However, because different communities of practice serve different purposes and manage different dependencies, they are now expanded as detailed below.</p>							
17	Tech lead forum ^{2,4}	Inter-team	<input checked="" type="checkbox"/>	✓	✓	Tech leads meet bi-weekly to share knowledge about technical issues and team architecture across teams, thus managing knowledge, process, and resource dependencies. Due to many participants, a large meeting room with many seats and audio-visual set-up is required.	Knowledge, resource
18	UX forum ^{New}	Inter-team	<input checked="" type="checkbox"/>	✓	✓	A bi-weekly forum for knowledge-sharing about UX. Primarily attended by members of the web and app teams, but others with an interest in UX can also attend.	Knowledge, resource

Inter-team stand-ups²: In paper 2, this referred to any kind of inter-team stand-up, however because different inter-team stand-ups may serve different purposes and manage different dependencies, they are expanded as detailed below.

19	Team leader stand-up ^{3,4}	Inter-team	☑	✓	✓	Weekly stand-up for sharing status across teams, with a focus on product, thus managing knowledge and resource dependencies. Conducted in open space.	Knowledge, resource
20	Product Owner stand-up meeting ⁴	Inter-team	☑	✓	✓	Replaced the product owner prioritization meeting in 2019. Similar in content to the prioritization meeting, but shorter and less detailed.	Knowledge, resource
21	Task force stand-ups ^{New}	Inter-team	☑	✓	✓	When task force teams were used, these held their own stand-up meetings. During these meetings, members of different teams get sit together (physically or digitally). Focus on product, thereby managing technical resource dependencies.	Knowledge, resource

Notes. In second column (coordination mechanism name), the superscript refers the dissertation papers in which the mechanisms have previously been reported: 1= Paper 1, 2= Paper 2, 3 = Paper 3, 4 = Paper 4, and New = not previously reported. TOPS characteristics are indicated by the following signs: ✓ and ☑, where ☑ refers to the primary characteristic, based on which type of dependency is primarily managed. The types of dependencies, following Strode, (2016), are described in Section 2.2.6.

4.1.2 Coordination roles

Coordination roles are performed by people, either individuals or teams, and contribute to managing inter-team dependencies by coordinating with others at an inter-team level. From the analyses, I identified thirteen unique coordination roles at Entur, ten individual and three team roles. All these roles have previously been reported in the research papers.

Nine roles operated at the inter-team level only, including all the three team roles (i.e., the test team, platform team, and task force team). For example, the test team (mechanism no. 11 in Table 9) performed testing of the products at an inter-team level, and also went into the different development teams as needed, assisting the teams with specific testing tasks. The test team thereby served to manage both activity process dependencies, in that the testing activity was required in order for the delivery to be considered done, and to entity resource dependencies, in that the resource (i.e., a tester) had to be available to the teams in order for the process to progress. The platform team (mechanism no. 12) worked in the same way, focusing on facilitating the development teams' technical environments. Additionally, three roles operated at the within-team level and one, the customer manager (no. 7), operated at the inter-organizational level. One role, the product owner (no. 1), operated both at the within-team and inter-team level. A more detailed example of this role is provided in Section 4.2.5.

In terms of dependencies, all thirteen roles are related to managing knowledge dependencies. Again, this may be explained by the high-level classification. Going into more detail on the dependency categories, many of these roles are related to the expertise knowledge dependency, where information about tasks is known only by certain persons or groups (Strode, 2016). Additionally, the historical dependency category applies to several of the roles, for example the program architects and project managers who had been with Entur for a long time and held knowledge about past decisions that were often important for coordination of tasks and decision-making across teams. Nine roles were related to process dependencies. Many of the manager roles coordinated business process dependencies, and the team roles were important for coordinating activity dependencies, in that certain activities had to be performed in certain orders for the development process to progress, as explained with the test team example above. Finally, all thirteen roles were related to resource dependencies, mostly the entity dependency in that often, the input of these roles were required across teams. Moreover, the platform team and the program architects (no. 8) were important for managing technical resource dependencies related to the software itself.

Table 9. List of all coordination roles observed during the fieldwork.

Coordination Roles							
Roles are coordination mechanisms performed by people coordinating with other people that contribute to managing inter-team dependencies.							
Coordination mechanism name	Organizational level	T	O	P	S	Description	Dependency Managed
1 Product Owner ⁴	Within-team, inter-team	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	Responsible for communicating their team's (or teams') prioritizations towards the clients, which can be different from other teams at the same time as they may depend on other teams to be completed.	Knowledge, process, resource
2 Tech Lead ⁴	Within-team	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	Responsible for the team's technical architecture (in collaboration with the program architects). Attends the tech lead forum and communicates team-level technical dependencies to other teams.	Knowledge, resource
3 Team leader ⁴	Within-team		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Responsible for the team members and team deliveries at an overall level. Primarily deals with the social aspect but are also involved in the development process.	Knowledge, process
4 Product manager ^{3,4}	Inter-team	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Responsible for POs, has overview of requirements and prioritizations across teams and clients. Involved in structural discussions.	Knowledge, resource, process
5 Development manager ^{3,4}	Inter-team	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Responsible for team leaders, has a high-level overview of teams' major tasks and prioritizations. Also responsible for staffing across the teams.	Knowledge, resource, process
6 Project manager ⁴	Inter-team	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Performed traditional project manager tasks during Enturs initial phases. The role was discontinued at the end of 2018.	Knowledge, process, resource
7 Customer manager ^{3,4}	Inter-organizational	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	One per major customer, this role is responsible for overall communication and coordination with clients, e.g., by attending meetings at the clients' sites. Brings information on e.g., requirements and specification back to the teams	Knowledge, resource
8 Program architects ^{3,4}	Inter-team	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	Concerned with the inter-team software, product, and organizational architecture. Involved in technical and structural discussions at all organizational levels.	Knowledge, resource, process
9 Agile method specialist ³	Inter-team	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	Responsible for agile methods and has overview of requirements, tasks, and prioritizations across teams. Facilitates retrospectives and other agile ceremonies across teams.	Knowledge, process, resource
10 Delivery process specialist ^{3,4}	Inter-team	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Implements an inter-team delivery process with the goal of aligning and improving inter-team product deliveries.	Knowledge, process, resource
11 Test team ^{2,3}	Inter-team	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Performs testing across teams and coordinate inter-team testing efforts. Some testing requires sitting with the development teams.	Process, resource
12 Platform team ^{2,3}	Inter-team	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Internal service team that manages technical resources by facilitating the teams' technical environment, providing a common platform. Some facilitation requires sitting with the development teams.	Knowledge, resource

13	Task force teams ^{2,3}	Inter-team			Temporary teams consisting of members from permanent teams used to implement interdependent features of high priority. The team is co-located while working together, and dissolves after feature completion.	Knowledge, resource
			☑	✓	✓	

Notes. In second column (coordination mechanism name), the superscript refers the dissertation papers in which the mechanisms have previously been reported: 1= Paper 1, 2= Paper 2, 3 = Paper 3, 4 = Paper 4, and New = not previously reported. TOPS characteristics are indicated by the following signs: ✓ and ☑, where ☑ refers to the primary characteristic, based on which type of dependency is primarily managed. The types of dependencies, following Strode, (2016), are described in Section 2.2.6.

4.1.3 Coordination tools and artifacts

The third coordination mechanism category in the taxonomy is coordination tools and artifacts, where tools manage dependencies by supporting the development process, while artifacts are by-products of the development process. At Entur, thirteen types of tools and artifacts were used at the inter-team level. Of these, eleven have been reported previously, while two, the service map (Table 10, mechanism no 12) and the team backlogs (no. 13) were identified in the new analyses. Team backlogs were included based on the insight that team-level mechanisms can manage inter-team dependencies, as further explained in the next section.

With respect to organizational level, twelve mechanisms operated at the inter-team level. For example, the shared backlog (no. 3), a digital artifact stored in the documentation tool JIRA, was designed specifically for tasks that involved more than one team and did not include any team-level tasks. Further, six mechanisms operated at the within-team level and two at the inter-organizational level. Of these, six operated at more than one level. For example, communication tools (no. 1) refer to several different communication tools, such as e-mail, Slack or Microsoft teams. All of these could be used either within teams, across teams, or across organizations, to coordinate inter-team issues. As another example, Entur used Objectives and Key Results (OKRs, no. 5) both within and across teams. OKR is a goal-setting framework where specific objectives, with accompanying key results, are formulated (Niven and Lamorte, 2016). OKRs operated at both organizational levels in that specific teams could formulate OKRs that dealt with dependencies to other teams, and the inter-team roles had their own sets of OKRs that operated at the inter-team level.

In terms of dependencies, all thirteen coordination tools and artifacts related to the broad category of knowledge dependency, as they all enabled the sharing of different types of knowledge and information. For example, the organization map (no. 4) provided information about who's who at Entur, thereby facilitating expertise knowledge coordination. Six of the mechanisms related to process dependencies, such as the inter-team delivery routines (no. 11) which provided an inter-team development process flow, thereby managing activity process dependencies. Finally, all thirteen tools and artifacts related to resource dependencies, either entity or technical. The above-mentioned organization map managed entity resource dependencies by enable people to locate each other across teams, but also the communication tools had the ability to search for people or teams, provided that the name of the entity was known. Moreover, technical resource dependencies were by managed by the coordination artifacts.

Table 10. List of all coordination tools and artifacts observed during the fieldwork.

Coordination Tools and Artifacts								
Tools manage dependencies by supporting the development process, while artifacts are by-products of the development process.								
Coordination mechanism name	Organizational level	T	O	P	S	Description	Dependency Managed	
1	Communication tools (e.g., Slack) ^{1,2,3,4}	Within-team, inter-team, inter-organizational	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	Tools such as e-mail, Slack or Microsoft Teams that enable digital communication and information sharing across teams (as well as within teams and across organizational boundaries) thereby managing resource (technical) and knowledge dependencies.	Knowledge, resource
2	Documentation tools ^{1,2,3,4}	Within-team, inter-team	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	Tools such as JIRA and Confluence, supporting the development process and enabling information sharing across teams, thus managing resource (technical) and knowledge dependencies.	Knowledge, resource
3	Shared backlog in JIRA ⁴	Inter-team	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	Digital artifact that supports knowledge sharing, in particular about development progress which contributes to managing process and resource dependencies.	Knowledge, process, resource
4	Organization map on confluence ²	Inter-team		<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	Digital artifact that provides information about the program members, primarily managing knowledge dependencies (who belong where, across teams)	Knowledge, resource
5	Objectives and Key Results ^{2,3,4}	Inter-team, within-team	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Conveys information on both technical (product) and organizational objectives and outcomes across teams, thus managing primarily resource, but also business process and knowledge dependencies.	Knowledge, resource, process
6	Burndown chart ³	Within-team, inter-team	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	A digital artifact that displays information related to completion of product-related development tasks and activities across teams.	Knowledge, resource
7	Prioritization document ³	Within-team, inter-team	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	A digital artifact that displays information on overall development priorities, across teams and clients. Enables communication and information sharing.	Knowledge, resource
8	Digital roadmap ³	Inter-team	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	A digital artifact that enables communication and information sharing related to overall product delivery milestones across teams, thus managing resource, process, and knowledge dependencies.	Knowledge resource, process
9	Physical roadmap ^{2,3}	Inter-team	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		A physical artifact. Has similar features as the digital roadmap, but is displayed in the open office space, thus containing less detail than the digital roadmap. People engage with it physically, e.g., by updating tasks.	Knowledge, resource, process
10	Task board ^{2,3}	Inter-team	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		Similar characteristics as the prioritization document but displayed in the open office space therefore showing top prioritizations only. People engage with it physically, e.g., by updating tasks.	Knowledge, resource
11	Inter-team delivery routines ^{2,4}	Inter-team	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		An inter-team delivery plan was gradually formed from 2019, including artifacts such as documents with shared routines	Knowledge, process, resource

				for deliveries, documentation and testing, and a common definition of done	
12	Service map ^{New}	Inter-team, inter-organizational	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	Provides information about Entur's services and products, which contributes to knowledge, technical resource, and business process dependencies between teams and also to clients.	Knowledge, process, resource
13	Team backlogs ^{New}	Within-team	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	Team-level artifact that can be accessed across teams, thereby managing knowledge and technical dependencies	Knowledge, resource

Notes. In second column (coordination mechanism name), the superscript refers the dissertation papers in which the mechanisms have previously been reported: 1= Paper 1, 2= Paper 2, 3 = Paper 3, 4 = Paper 4, and New = not previously reported. TOPS characteristics are indicated by the following signs: ✓ and , where refers to the primary characteristic, based on which type of dependency is primarily managed. The types of dependencies, following Strode, (2016), are described in Section 2.2.6.

4.1.4 New insights on previous analyses

Throughout this thesis project, my understanding of coordination mechanisms has developed and matured. Accordingly, the new analysis led to some new insights, which I will present in the following.

1) A team-level or inter-organizational level mechanism can manage inter-team dependencies.

The first new insight was that a coordination mechanism does not need to be located at the inter-team level to manage inter-team dependencies. In Paper 3, only mechanisms that operated strictly at the inter-team level were included. However, re-examining the material led me to understand that coordination mechanisms at the team and intra-organizational levels could also contribute to managing dependencies at the inter-team level.

Several coordination mechanisms, for example, stand-up meetings, backlogs, and Slack were used across organizational levels. It, therefore, made sense to include these. I further included all team-level and inter-organizational level mechanisms that fit the definition of inter-team coordination mechanisms as a ‘process, entity or arrangement that contributes to managing inter-team dependencies’ (see Figure 14). For instance, dependencies to other teams were often discussed at team-level retrospectives, which led to actions being taken that contributed to resolving inter-team dependencies. The following example is taken from my field notes: *During a team leader retrospective in October 2018, the team leaders discussed how Slack could be used for optimal inter-team communication. Some team leaders share negative experiences with people taking too long to respond, which could cause delays. The team leaders discussed this for a while and concluded that all team leaders were to bring the message back to the teams and encourage people to reflect on when and how often they could respond* [Field notes, October 2018]. Similarly, the client meetings (inter-organizational meetings) manage

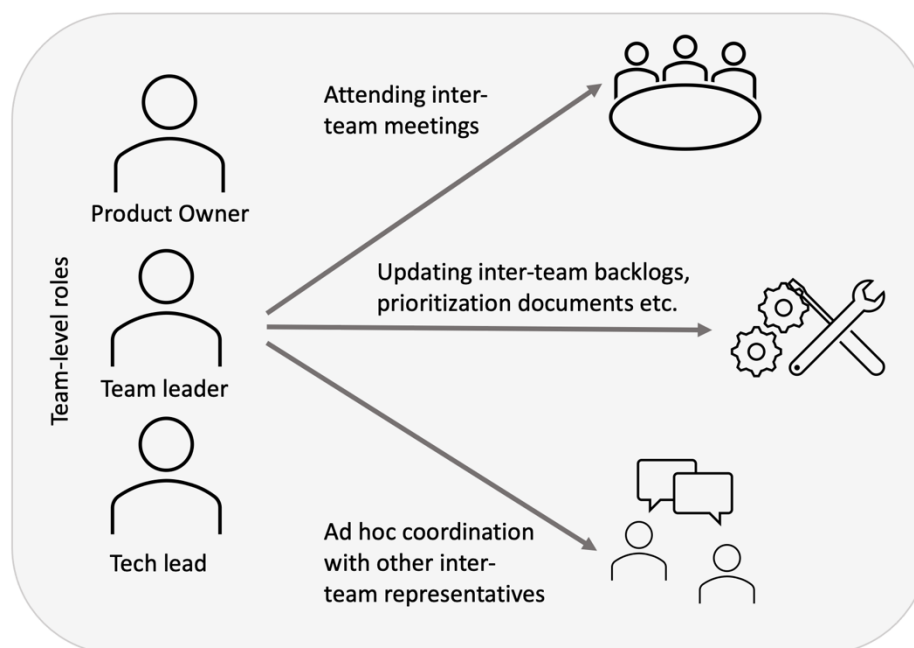


Figure 15. A team-level role operates as an inter-team coordination mechanism.

inter-team dependencies as the topics discussed were focused on the software delivery as a whole.

As another example, the product owner and team leader roles were located at the team level in the organizational matrix (see Figure 9). As illustrated by Figure 15, they contributed to inter-team coordination by sharing knowledge and information across teams and bringing information back to their respective teams. “*The team members get information about other teams’ tasks and priorities through me or my team leader. I have a good overview of priorities from the business side, and he has details about what everyone is doing everywhere. [...] We share a lot ‘on the go’, but also during our weekly team meetings.*” [I01, Product owner]. Similarly, the tech lead role participated in the inter-team tech lead forum, thereby contributing to managing inter-team dependencies by sharing goals and knowledge across teams. One tech lead defined the role in the following way that illustrates the inter-team focus of the role: “*In Entur, the tech lead role is all about technical coordination, and in a way, to be a person that has a foot within the team and another outside the team and in the other teams*” [I22, Tech lead].

2) Certain mechanisms can be collapsed into overall types, while others should be split into unique mechanisms.

In the analyses for Paper 3, several judgment calls were made as to what to include as specific or unique mechanisms. For example, I decided to collapse all communication tools into one overall mechanism. This was done to make the taxonomy applicable; not all companies use Slack as the primary communication tool, and Entur used several other tools as needed. For example, they used Microsoft Teams if required by clients or other collaborators. I did not have access to Microsoft Teams in the same way as I had with Slack, which limited the knowledge I gained about how these alternative mechanisms were used. Nevertheless, because other large-scale programs may use other tools or meetings, keeping the broader categories of *communication tools* and *documentation tools* appears meaningful. Following the same logic, I now include *inter-team delivery routines* consisting of shared delivery routines, shared

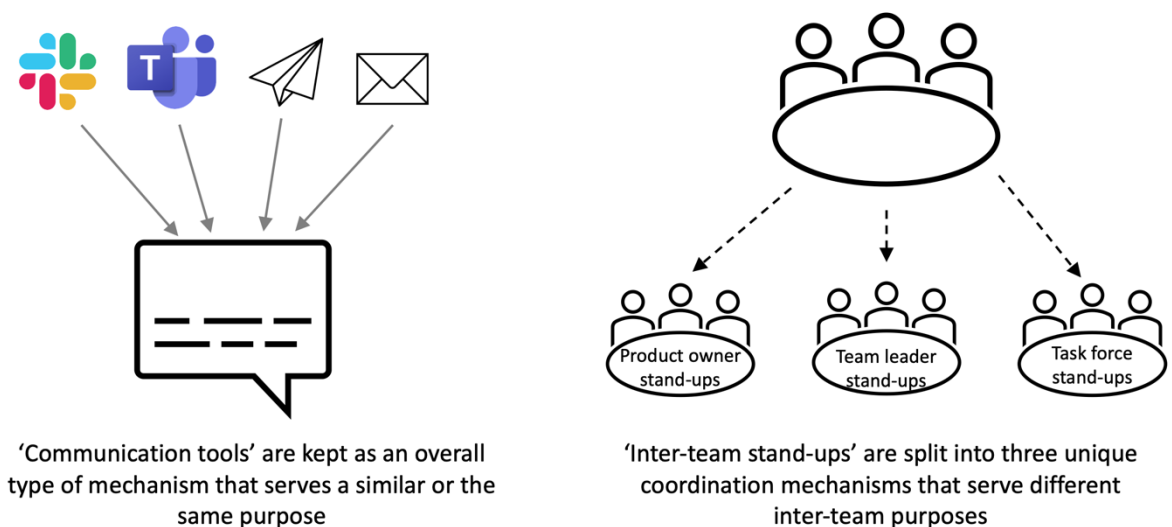


Figure 16. Examples of categorizations of coordination mechanisms.

documentation and testing routines, and a common definition of done as an overall mechanism, which was reported as separate mechanisms in papers 2 and 4 (see Table 10).

On the other hand, as seen in Table 8, I decided to split *inter-team stand-up meetings* and *inter-team retrospectives* into the unique mechanisms they were composed of (i.e., product owner-, team leader-, and task force stand-up meetings and product owner-, team leader-, and tech lead retrospectives, and). Similarly, communities of practice, including the tech lead forum and UX forum, were reported as one mechanism in Paper 3. During the present analysis, I concluded that all these meetings should be reported as unique inter-team coordination mechanisms because they serve different purposes and, therefore, manage different dependencies. For example, the product owner stand-up meeting focused more on client prioritizations and business process dependencies, whereas the team leader stand-up is typically more oriented towards technical product dependencies or knowledge dependencies across teams. Figure 16 illustrates the two alternative decisions made.

3) Excluding elements compared to the original papers.

Finally, as my understanding of inter-team coordination has evolved, eight elements were previously included as coordination mechanisms I no longer consider as such, in light of the definition of coordination mechanisms adopted for this thesis (i.e., processes, entities, and arrangements that contribute to managing dependencies). These were re-framed or removed altogether (see Table 11). In the following, I briefly comment on each of them.

First, *internal team practices* are too vague, and it is not clear what it contains. In Paper 1, we use this to refer to activities conducted by the product owner and the team, such as team retrospectives or stand-up meetings, from which the product owner brings relevant input back to the inter-team level. Subsequent analyses have refined my understanding of how team-level practices contribute to inter-team dependency management, as detailed above in the first point made in this section. I now include team-level stand-up meetings, team-level status meetings, and team-level retrospectives as independent team-level coordination mechanisms that manage inter-team dependencies.

Second, in Paper 2, where we applied the theory of coordination for agile development teams (Strode 2012) at the inter-team level, we included coordination mechanisms such as shared routines, having a common definition of done, and taking on other teams' tasks. Based on my current understanding, these are best considered as practices and artifacts that are part of the overall coordination mechanism 'inter-team delivery routines' presented in the above section and Table 10 (mechanism no. 11).

Moreover, in Paper 2, using Strode's (2012) theoretical framework, we included open office space and co-location as coordination mechanisms. Since then, insights gained during later analyses have led me to no longer consider these as coordination mechanisms. Indeed, both co-location and open office spaces enable coordination, but these elements do not *in themselves* manage dependencies. I return to this point in the discussion section.

Table 11. Excluded elements.

Elements considered coordination mechanisms in the papers that are removed from the overall list.		
Coordination mechanism name	Organizational level	Description and reason from removal
1 Shared routines for deliveries, documentation, and testing ²	Inter-team	These are now included under the coordination tool no. 11 'inter-team delivery routines' as one overall group of mechanism.
2 Common definition of done ²	Inter-team	This is now included under the coordination tool no. 11 inter-team delivery plan as one overall group of mechanism.
3 Taking on other teams' tasks ²	Inter-team	This element does not appear to fit in the taxonomy. While taking on other teams' tasks has value from a coordination point of view, it does not fit the definition adopted for this thesis.
4 Internal team practices ¹	Within-team	Too vague to be considered a coordination mechanism in the definition adopted for this thesis. Consists of more than one element, and is covered by the team-level coordination meetings listed above
5 Co-location ²	-	Enables coordination rather than being a coordination mechanism on its own.
6 Open office space ²	-	Enables coordination rather than being a coordination mechanism on its own.
7 Inter-team status meetings ²	Inter-team	This element consists of several coordination meetings: no. 19 'team-leader stand-up meetings', and no. 3, 'product owner prioritization meetings', and it is therefore deleted.
8 Temporary team arrangements ²	Inter-team	This refers to the same as coordination role no. 13 'task force teams', and it is therefore deleted.

Notes. In second column (coordination mechanism name), the superscript refers the dissertation papers in which the mechanisms have previously been reported: 1= Paper 1, 2= Paper 2, 3 = Paper 3, 4 = Paper 4, and New = not previously reported.

4.2 RQ2: How are coordination mechanisms used to manage dependencies in large-scale agile software development?

The second research question seeks to answer how coordination mechanisms are used to manage dependencies. The analyses conducted in this research project has led to the realization that, in order to understand how a coordination mechanism manages dependencies, we need to look at not only what the mechanism does or how it is used, but also its underlying technical, organizational, physical, and social (TOPS) characteristics. This topic was the focus of Paper 3, where the TOPS framework was developed based on the analysis of all the inter-team level data material collected from the fieldwork. In the new analyses conducted for this thesis summary, I have expanded the analysis to include material collected at all organizational levels and used the TOPS framework to characterize all 47 mechanisms listed in tables 8-10.

In this section, I first present the TOPS framework, before turning to describing how coordination meetings, roles, and tools and artifacts are used to manage dependencies in large-scale agile using examples from the data in tables 8-10. Following this, I provide three detailed examples on how coordination mechanisms are used to manage dependencies.

4.2.1 The TOPS framework

The TOPS framework describes the underlying characteristics of coordination mechanisms and how they relate to dependency management. The notion that coordination mechanisms have underlying characteristics has its roots in ideas of software development as a socio-technical activity that requires the use of mechanisms with a socio-technical character. However, we found that these two characteristics were too narrow to capture the full picture that additional

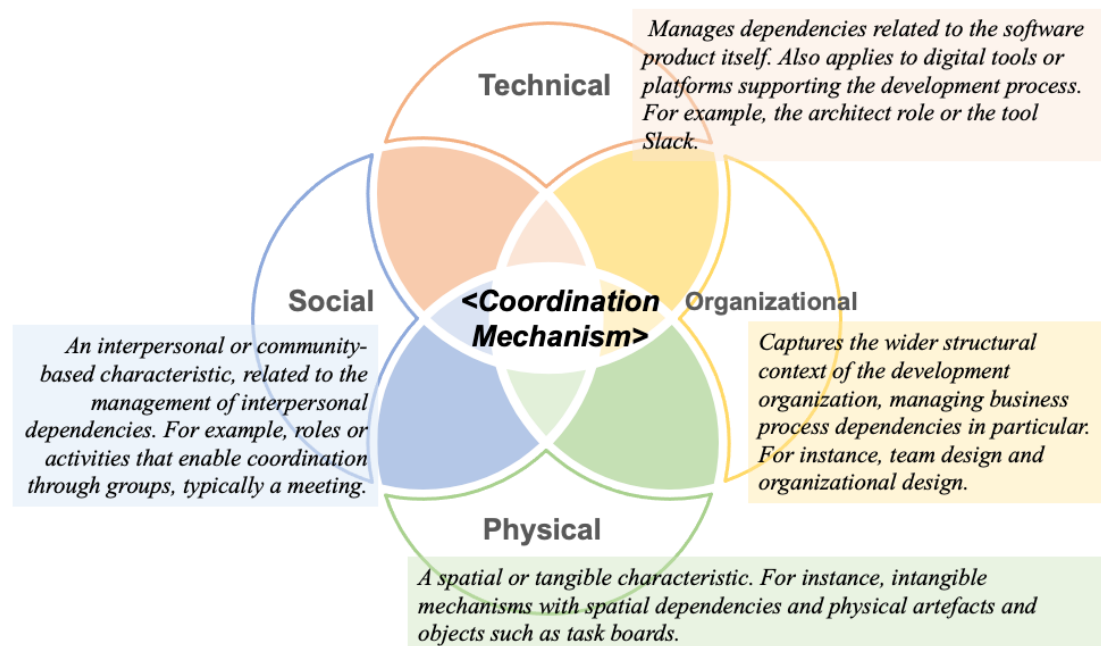


Figure 17. The TOPS framework.

characteristics are needed in order to capture and explain the complexities of modern large-scale software development organizations. Accordingly, the analysis for Paper 3 resulted in the description of four different characteristics, that is, the TOPS characteristics.

First, *technical* refers to characteristics of the coordination mechanism that manages dependencies related to the software product itself. This dimension also applies to digital tools or platforms supporting the development process—for example, the program architect role (Table 9, no. 8) or the tool Slack (Table 10, no. 1).

Second, *organizational* includes characteristics of the coordination mechanism that captures the wider structural context of the development organization, managing business process dependencies in particular. For instance, aspect of the coordination mechanism that relates to team design and organizational design. The program management roles (Table 9, no. 4-6) are examples of coordination mechanisms that display the organizational characteristic.

Third, *physical* refers to the spatial or tangible characteristics of the coordination mechanism. For instance, intangible mechanisms with spatial dependencies and physical artifacts and objects such as task boards (e.g., Table 10, no. 9 and 11).

Fourth, the *social* characteristic refers to the interpersonal or community-based characteristic of the coordination mechanism and is related to the management of interpersonal dependencies. For example, roles or activities that enable coordination through groups, typically a meeting. Together, these elements form the TOPS framework, which can be applied to analyze coordination mechanisms to learn more about how they contribute to coordination by managing dependencies. Figure 17 displays the TOPS characteristics.

Tables 8-10 provide summarizing information on all the coordination mechanisms. The T, O, P, and S columns displays which of the characteristics apply to each mechanism. Here, the ✓ symbol indicates that the characteristic is present as a secondary characteristic, whereas a ☑ symbol indicated the primary characteristic. This characterization is guided by how the mechanisms are represented by the data. In the tables, the “Description” column provides brief textual details on how each of the mechanisms manage dependencies.

4.2.2 The TOPS characteristics of coordination meetings

The *technical* characteristic of coordination mechanisms is related to how the mechanisms manages dependencies related to the software product. Examining the TOPS characteristics of the coordination meetings in Table 8, fourteen of the meetings have the technical as the primary characteristic. Additionally, six meetings have technical as a secondary characteristic. In a large-scale agile software development setting, it is not surprising that almost all coordination meetings have a technical characteristic, as most meetings were related to coordination of dependencies related to the technical software development. given the nature of the product development at hand.

As a contrast, the *organizational* characteristic was found in only six of 21 coordination meetings. No meetings have this as the primary characteristic. The low number of meetings with the organizational characteristic makes sense, as the development teams focused on software development rather than organizational and structural aspects. At some meetings, however, such as manager-level meetings and many of the product owner meetings, it made sense to discuss aspects of the team organization and wider structural issues. Further, as can be

seen in Table 8, all meetings that held the organizational characteristic serve to manage process dependencies.

The *physical* characteristic was assigned as a secondary characteristic to all meetings. This is not to say that all meetings were exclusively co-located, or that physical presence is a firm requirement for any of these meetings. Rather, the data indicated that physical aspects had an impact on the meetings. For example, in terms of appropriate size of meeting rooms relative to the number of participants, air quality during long meetings (which was notably felt during some team leader retrospectives), and the quality of audio-visual set-up during hybrid meetings.

Finally, all meetings displayed the *social* characteristic, with seven meetings displaying this as the primary characteristic. As meetings are, per definition, social encounters between people this characteristic is evident.

4.2.3 The TOPS characteristics of coordination roles

In the large-scale development setting at Entur, all coordination were involved with the software product one way or another, and as such, displayed the *technical* characteristic. Of these, seven of the roles were more directly involved with the software development and held technical as the primary characteristic. This pertained to, for example the tech leads (Table 9, mechanisms no. 2) and the program architects (no. 8) who were technical experts at the team-level or program level. These individuals often worked directly with coordinating technical dependencies across teams.

Further, eight coordination roles held the *organizational* characteristic in that they worked with coordination of process dependencies and structural issues across teams. Of these, four coordination roles, the program managers (mechanisms no. 4-6), as well as the delivery process specialist (no. 10) held the organizational as the primary characteristic, as these roles had the team organization and program organization at the heart of their roles.

The *physical* characteristic related to the coordination roles whose physical presence were important for efficient inter-team coordination. This applied to four of the roles (albeit as secondary characteristics only), namely the customer manager (no. 7) who would attend client on-site meetings, and the test, platform, and task force teams (no. 11-13). As described in Section 4.1.2, the representatives from the test and platform teams would sit with the development teams as needed. The task force teams were temporary teams consisting of members from the different development teams who worked together in temporary constellations related to specific product implementations. They were co-located while working together, which makes the physical characteristic evident.

Finally, being performed by humans, the *social* characteristic applied to all roles as they were all being performed by people coordinating with other people to manage knowledge dependencies. However, two roles, the team leader (no. 2) and the agile method specialist (no. 9), were characterized as primarily social roles. At Entur, the team leaders were responsible for the team members and at an overall level and dealt with the inter-personal and social aspects of the team. At the inter-team level, team leaders coordinated with each other to keep each other informed of any team-level issues that could affect other teams. The agile methods specialist worked at an inter-team level, facilitating inter-team retrospectives and other agile ceremonies, thereby facilitating and coordinating social and inter-personal interaction at an inter-team level.

4.2.4 The TOPS characteristics of coordination tools and artifacts

As tools and artifacts per definition are used during software development to support the process, their primary characteristic was *technical*. The only exception was the organization map (Table 10, mechanism no. 4), where the primary characteristics was the social as its function was to show the people of Entur.

The *organizational* characteristic applied as a secondary characteristic to four of the coordination tools and artifacts. One of these, the service map (no. 12), was included as a new mechanism during the final analysis. This artifact provided information about Entur's services and products and was available both internally and externally to the organization. The organizational characteristic is evident in the information it provides on Entur's business, thereby contributing to coordination of business process dependencies not only across teams, but also in a wider, inter-organizational setting.

Three coordination tools and artefacts were assigned the *physical* characteristic. These were the OKRs (no. 5), because they were developed during co-located OKR workshops, and also often were displayed physically for easy overview (i.e., on post-its or similar), and the physical roadmaps and task boards that hung in the office spaces (see, for example Figure 1 in Paper 1). All these were secondary characteristics.

Finally, twelve of the tools and artefacts were characterized with the *social* as a secondary characteristic, as they all contributed to managing knowledge dependencies. Additionally, the organization map (no. 4) was considered primarily social, as described above.

4.2.5 Three detailed examples on how coordination mechanisms are used to manage dependencies.

In the following, I present three in-depth examples of how coordination mechanisms were used to manage dependencies at Entur. These examples, the tech lead forum, the product owner role, and the communication tool Slack, were selected because they were important mechanisms at Entur that were often in focus during fieldwork and interviews. During the 1.5 years, I observed seven tech lead forum meetings and more than 20 product owner meetings. Furthermore, I interviewed four tech leads and ten product owners. I also collected and analyzed material from Slack channels and discussed Slack use with people during most interviews. For each example, I describe the coordination mechanism's TOPS characteristics to explain which dependencies are managed, and how. I also describe how the mechanisms have changed over time.

1) The tech lead forum is categorized as a scheduled coordination meeting in the taxonomy of inter-team coordination mechanisms (see Figure 14). At Entur, the meeting was modeled after the ideal of 'Spotify guilds' or communities of practice meetings and was aimed at sharing architecture-related knowledge and providing an overview of technical dependencies across all the teams. Participants were the tech leads from each development team, one or more of the program architects, and other interested stakeholders such as the product and development managers. Figure 18 shows a screenshot of a meeting invitation to the tech lead forum that

states the purpose and desired outcomes of the forum. The text in Figure 18 has been translated from Norwegian to English.

In terms of TOPS characteristics, the meeting consists of technical, social, and physical characteristics. *Technical*, because of the focus on discussing and resolving technical issues across teams which contributes to managing technical resource dependencies. Social, because of the interpersonal aspect. Not only did the meeting contribute to managing knowledge dependencies, including expertise and historical dependencies by having key expert roles in the same meetings, it also contributed to shared knowledge and shared goals across teams as topics were discussed and decisions were made. The forum further contributed to fostering a sense of mutual respect among the tech leads by encouraging everyone to discuss and contribute. Further, the *physical* characteristic is evident because of the number of people attending the meeting. Over time, there were so many participants that the largest meeting room was not big enough, and additional chairs and an audio-visual set-up were required to ensure that everyone who wished could attend. However, the tech lead forum did not deal with any structural or organizational matters, therefore the *organizational* characteristic is not relevant.

The forum can be considered part of a coordination strategy for managing technical dependencies in combination with mechanisms such as a platform team, measurement and tracking tools, and artifacts such as OKRs, and task force teams. Finally, the mechanism was adaptable in response to changing coordination needs. Over time, we observed how the forum was subject to change, both subtly through the focus on continuous improvement (e.g., by adding or removing agenda points or holding retrospectives to learn and improve) and more directly evident changes resulting from a need to adapt to external changes (such as the need to make the meeting digital during the pandemic).

Purpose

The Tech-Lead forum aims to establish technology and architecture management across teams and create an arena for knowledge sharing and to exchange experiences.

Furthermore, the purpose is to establish a forum for technical decisions and joint best practices and guidelines in terms of technology and solution architecture.

Agenda

An overview of the agenda and decisions can be found at <https://exampleurl/wiki/spaces/AR/pages/Tech-Lead+Forum+-+Agenda+and+decision+log>

Purpose and desired result:

- Establish better communication between the teams
- Get a better overall understanding and overview of what is happening and how the systems fit together
- Get better control and quality of the architecture and solutions
- Achieve a higher speed of development and better quality

Figure 18. Meeting invitation to the tech lead forum (translated from Norwegian).

2) The product owner is categorized as an *individual coordination role* in the taxonomy. At Entur, the product owner was considered part of the development teams. Most product owners belonged to one team, but some were responsible for two or three teams (see Figure 9). Their background was varied, with some product owners holding technical engineering degrees while others came from different industries, such as marketing and business development. Some had much domain experience, while others were newcomers.

The product owner role corresponded with the delivery areas of the software product, and each product owner was responsible for the prioritizations for that delivery area and for coordinating priorities towards the overall product, thereby contributing to managing technical dependencies. They were also responsible for communicating with clients and gaining an overview of their needs in collaboration with the customer responsible. This contributed to managing business process dependencies. Moreover, they were important for managing knowledge dependencies as one of the team roles that performed much inter-team coordination alongside and in collaboration with team leaders and tech leads, as illustrated by the quotes in Section 4.1.

In terms of the TOPS characteristics, this mechanism displays strong technical characteristics because the role primarily deals with technical dependency management. Further, as a role being performed by humans coordinating with other humans to share information and knowledge, the *social* characteristic is evident. Additionally, the *organizational* characteristic applies to some extent, as the product owners could be involved in structural discussions in light of their focus on how to optimize the prioritization of (their part of) the product across the large-scale development program. As such, the product owner role was part of a coordination strategy for task prioritization.

The product owner role at Entur was a much-discussed role that was subject to change (which I discuss in more detail in Paper 4). Within their own fora, including their weekly meetings and during retrospectives, the product owners worked on adjusting and improving how they worked with knowledge sharing and how they collaborated towards shared goals, and they improved their communication patterns.

3) Slack is a *digital communication tool* that enables and supports coordination among individuals, groups, and organizations. While several digital communication options were available at Entur, Slack was allegedly by far the most used communication platform. Slack allows users to communicate in public or private group channels as well as with private direct messages (Stray & Moe, 2020). Each team had their own channel, the same had the various team leaders, the product owners, and the tech leads. There were also specific channels for inter-team coordination related to specific tasks, technologies, clients, and so forth. Figure 19 displays a screenshot from an open Slack channel.

Slack contributed to the effectivization of knowledge sharing and managing knowledge dependencies by enabling swift and timely communication. For example, rather than waiting for a particular role to be available in person, team members could send a message and often get a timely response. Slack also allows for video chats, file sharing, and the set-up of bots for the automation of various tasks. At Entur, the use of bots contributed to managing, for instance,

expertise knowledge dependencies as specific test tasks could be automated, thereby saving the testers time and effort.

Using the TOPS framework to characterize this coordination mechanism, the communication tool is primarily *technical* as it was first and foremost used to resolve technical dependencies, and most of the Slack communication was product focused. However, Slack also has a strong *social* characteristic in that it can be used to connect people, in particular when people are working from different locations. Furthermore, in addition to product-focused communication, people also used it for more social conversations. Over time, I observed how Slack use changed. For example, after a retrospective, the product owners decided to adapt the use of their Slack channel (see Paper 1 for more details). As another example, in 2019, Entur implemented channel guidelines as the number of clients and similar channel names grew (see Paper 4). Summing up, these examples and many other observations of Slack use at Entur (i.e., the many conversations across many channels and the many bots used) supported that the tool is versatile and widely applicable. It can therefore be considered as part of any coordination strategy.

4.3 RQ3: How can we analyze coordination mechanisms in large-scale agile?

In response to the third research question of this thesis, I introduce a framework for analyzing coordination mechanisms. This framework, named the Framework for Analyzing Large-scale agile Coordination mechanisms (FALC), is based on the answers to the previous research questions and the contributions from the papers. The framework outlines a four-step approach to analyzing coordination mechanisms in agile: 1) identifying coordination mechanisms, 2)

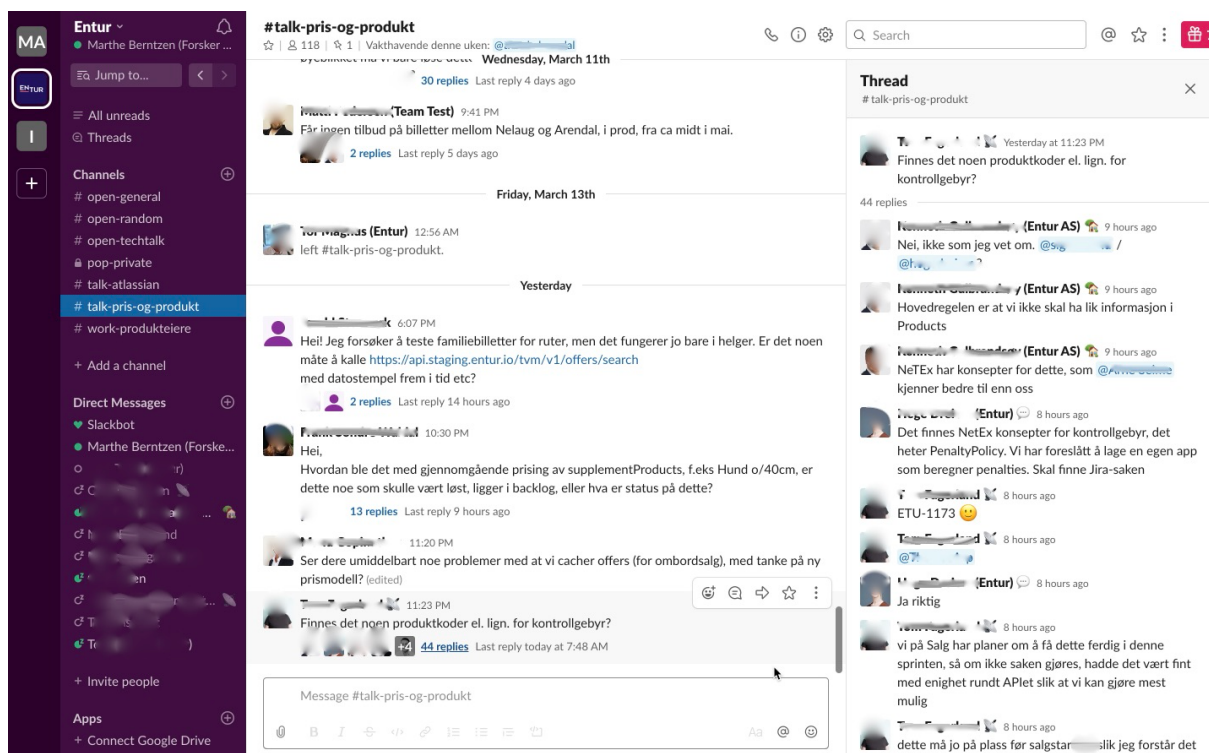


Figure 19. A screenshot of a Slack channel.

mapping the mechanisms' TOPS characteristics, 3) map how each mechanism manages dependencies, and 4) analyze change in the environment and the mechanisms themselves. Based on the information gained from these steps, coordination strategies can be formed.

FALC provides a holistic approach to analyzing and understanding coordination mechanisms within any large-scale agile organization. The framework is intended to provide awareness of and insight into the specific coordination situation at hand and is meant to be easily tailored to the organizational context under study.

From a practical perspective, the knowledge generated from applying FALC is intended as input for making informed decisions about which coordination mechanisms to use to manage the dependencies specific to the large-scale development organization, project, or program. FALC provides guidance for collecting and analyzing data on coordination mechanisms, and it is designed to be useful to researchers and practitioners alike. FALC is illustrated in Figure 20, and Table 9 presents a step-by-step overview of how the framework can be applied.

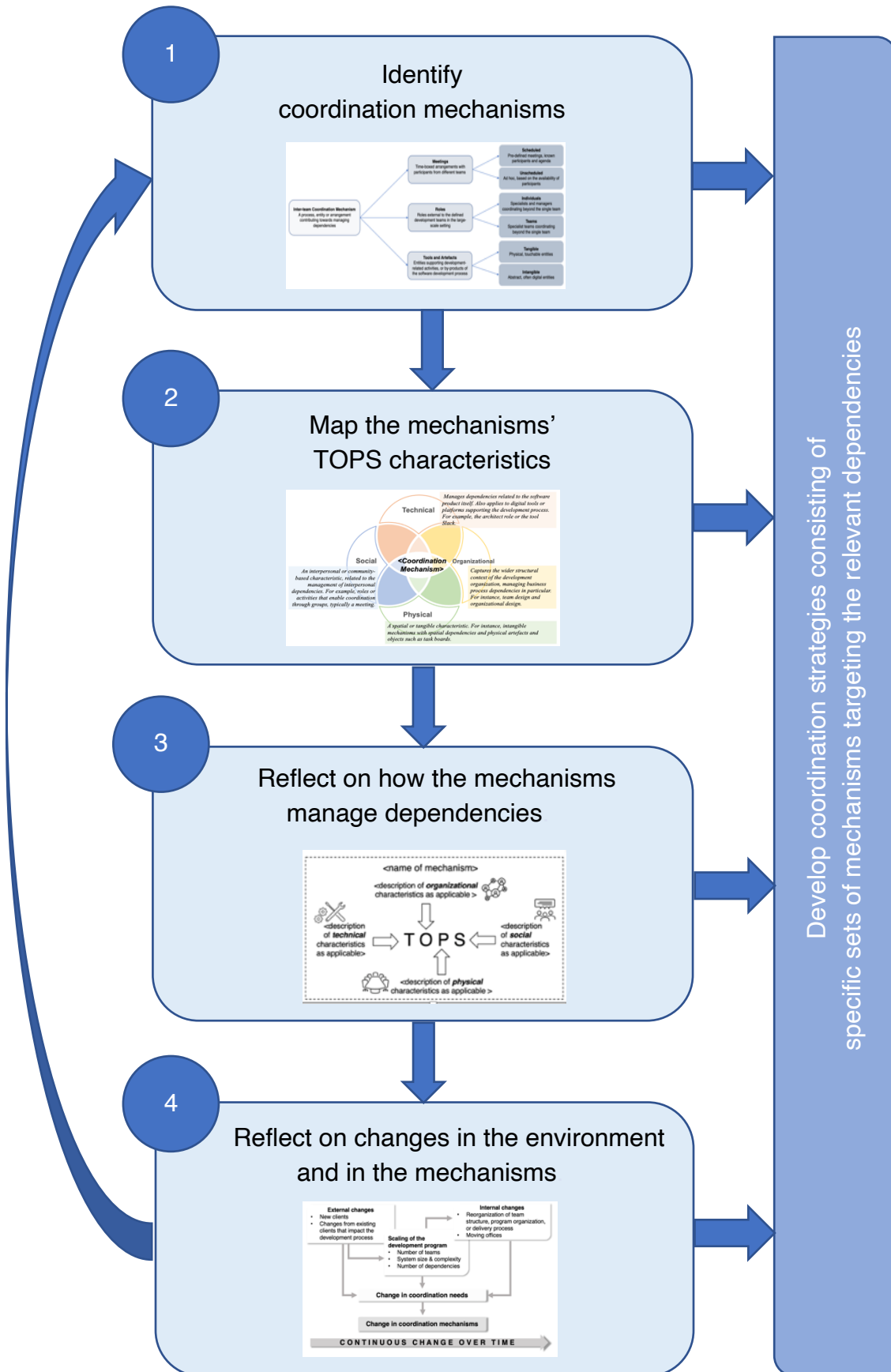


Figure 20. The Framework for Analyzing Large-scale agile Coordination mechanisms (FALC).

4.3.1 Applying FALC

In this section, I present the contents of the framework and describe how it is intended to be applied. For each step, a set of questions intended to guide the analysis is included in Table 9.

Step 1. Identify coordination mechanisms.

The first step involves gaining an overview of which coordination mechanisms are currently in use by identifying and describing each individual mechanism. Here, the taxonomy inter-team coordination mechanisms (Figure 14) provide a tool for structuring the analysis. Mapping each mechanism identified according to the categories of the taxonomy provides an organized overview of the scene that alleviates subsequent analyses. The product generated from this step is a list of all meetings, roles, and tools and artifacts used in the organization. The next two steps of the analysis involve understanding how the mechanisms manage dependencies.

Step 2. Mapping the technical, organizational, physical, and social characteristics.

When all coordination mechanisms have been mapped, the next step in the framework involves gaining a deeper knowledge of the mechanisms by analyzing their underlying TOPS characteristics. This analysis serves as input for understanding how each mechanism contributes to managing dependencies. The descriptions for each of the TOPS dimensions are provided in Figure 17. In Paper 3, I developed the TOPS visual template (Figure 21) that is aimed at supporting coordination mechanism characterization. The information gained from this step is meant to serve as input for the third step, understanding dependency management.

Step 3. Understanding dependency management.

During this part of the analysis, the objective is to reflect on what types of dependencies are present in the large-scale agile environment and how the mechanisms contribute to managing dependencies. FALC is not prescriptive in terms of how to define dependencies, but I recommend using the categories of knowledge, resource, and process dependencies outlined in the dependency taxonomy (Strode, 2016), as these categories were developed from an agile

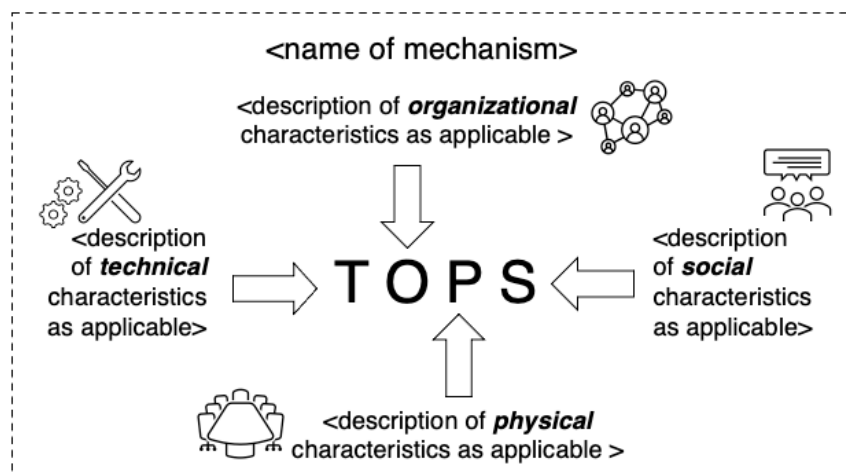


Figure 21. The TOPS visual template (From Paper 3)

context. During this step, it can also be worthwhile reflecting on how the mechanisms contribute to relational coordination by supporting shared knowledge and shared goals, or how they facilitate high-quality communication.

Step 4. Reflecting on change in coordination mechanisms.

As a fourth step in the analysis, I suggest reflecting on change. Reflecting on how past internal and external change events have shaped the coordination process at the organization can lead to interesting insights on which and how coordination mechanisms are currently used. Reflecting on the more subtle changes that happen on a day-to-day basis because of continuous improvement provides additional insight.

In Paper 4, I proposed a model that describes how coordination mechanisms change over time in response to changes in the large-scale organization's internal and external environment and as part of the organization's ability to respond to change and continuously improve, which are also core aspects of agile. Applying the model of change in coordination mechanisms provides insights into past and current coordination needs. The model, which is shown in Figure 22, can therefore be used as a guide for this part of the analysis.

Step 5. Summary and forming coordination strategies.

In the fifth and final step, the analysis is summarized. Completing the above steps should result in a detailed list of which coordination mechanisms are used, what they comprise, and how these elements manage different dependencies in the large-scale situation. Together, these elements generate insights that can be used 1) to generate empirical knowledge to inform research or 2) to form coordination strategies to increase coordination effectiveness in practice. Coordination strategies are sets of coordination mechanisms that address specific dependencies (Strode, 2012). The insight gained from using FALC can provide input to overall coordination strategies and specify which mechanisms they include. An example strategy, taken from Paper

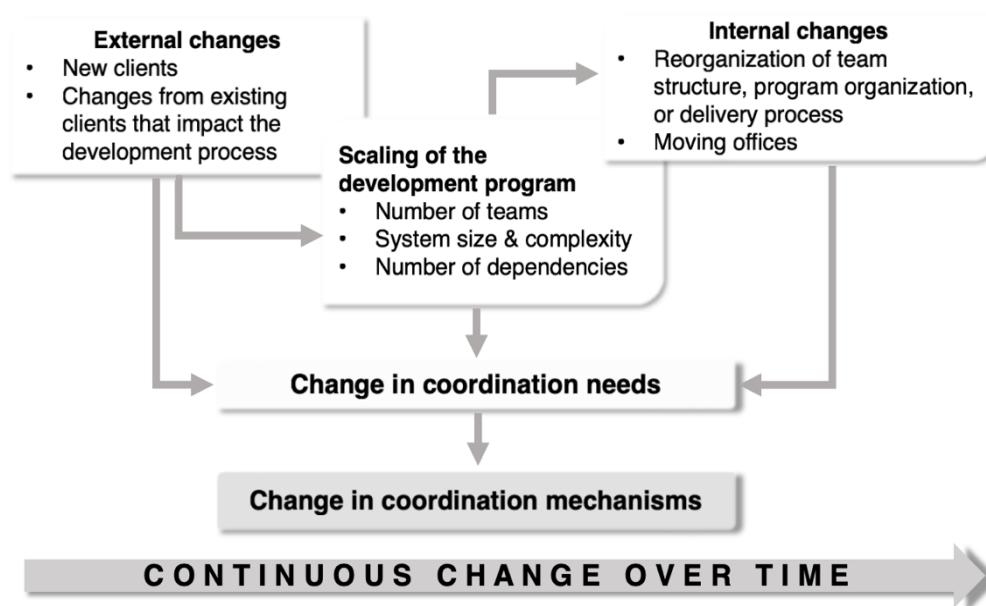


Figure 22. A model of change in coordination mechanisms over time.

2, for gaining and maintaining overview across teams can consist of coordination meetings such as inter-team stand-ups and status meetings for coordination roles such as product owners and team leaders, and having inter-team product demonstrations for all program participants, and use coordination tools such as Slack across teams, and shared backlogs, roadmaps and organization maps to support knowledge sharing and easy orientation in the large-scale organization. Other example strategies are presented in the detailed examples in Section 4.2.5

4.3.2 A flexible and adaptable framework

FALC is intended as a flexible, adaptable, and non-normative framework. The goal is that it can be adapted for use in most large-scale agile contexts with as few prescriptions and instructions as possible.

As indicated by the arrows between boxes 1-4 in Figure 20, the five steps are intended to be carried out in a rather sequential order for optimal results. However, the framework is flexible, such that, for example, skipping a step is possible. It is also possible to stop the analysis at any step and proceed to formulate coordination strategies, as indicated by the downward pointing arrows from each of the first four boxes to box 5. An alternative way to use FALC is to start by first formulating the desired coordination strategies and then carrying out steps 1-4 to identify which mechanisms can be included in the desired coordination strategies. For instance, the goal could be to form a coordination strategy for prioritizing tasks across teams. Knowing this, it is possible to start with step 1, identify which coordination mechanisms need to be in place, and move forward from there. Perhaps some mechanisms are already available, whereas others must be added or changed to manage a given set of dependencies.

Because the framework is context-sensitive and non-normative, it can be applied by researchers and practitioners. It is designed such that the steps can be carried out by a single individual or by groups during a workshop. Depending on the size of the large-scale development setting, the analysis is likely to be time-consuming. I, therefore, recommend running a small pilot session and/or splitting the steps into separate sessions. Another alternative, if little time is available, is to focus on one category of coordination mechanisms at a time.

Table 12. A practical guideline for using the Framework for Analyzing Large-scale agile Coordination mechanisms.

STEP	HOW-TO	QUESTIONS TO SUPPORT THE ANALYSIS
Step 1. Identify coordination mechanisms	Use the taxonomy of inter-team coordination mechanisms (Figure 14) to create a list of which mechanisms are in use. Provide short descriptions of what they do or how they are used.	<ul style="list-style-type: none"> - Which meetings are used to coordinate between teams? - Which roles deals primarily with coordination between teams? - Which tools are used to between teams? - Which artifacts? - Which organizational level(s) applies to the different mechanisms?
Step 2. Understand their characteristics	Use the TOPS framework and visual template (Figure 17s and 21) to map the underlying characteristics of the mechanisms.	<ul style="list-style-type: none"> - What makes this a technical coordination mechanism? - What social characteristics are present? - Are there any physical characteristics? - What organizational characteristics are evident? - What seems to be the primary TOPS characteristics?
Step 3. Understand how they manage dependencies	Reflect on what types of dependencies are present in the large-scale agile environment and how the mechanisms contribute to manage dependencies. Using the elements of relational coordination (Gittell, 2006) and Strode's (2016) dependency categories is recommended, but not required.	<ul style="list-style-type: none"> - Which dependencies does each of the mechanisms manage? - How does the mechanisms contribute to shared knowledge and shared goals, or facilitate high-quality communication? - Are there any dependencies that are not (sufficiently) managed? - If so, what is needed for efficient dependency management? Are more or other coordination mechanisms needed?
Step 4. Identify and reflect on changes in coordination mechanisms	Reflect on past and present change events as well as more subtle changes that happen day-to-day in the environment. Using the model of change in coordination mechanisms (Figure 22) is recommended.	<ul style="list-style-type: none"> - Which past external and internal changes have impacted coordination? - Which changes are happening at the moment and how do these changes influence coordination? - Do we know about any upcoming changes that will influence coordination? - What changes can be made to existing coordination mechanisms to meet these needs? - Will any mechanisms need to be adjusted, removed, or replaced?
Step 5. Develop coordination strategies	Summarize the analysis and formulate coordination strategies consisting of sets of coordination mechanisms that address specific dependencies or coordination needs.	<ul style="list-style-type: none"> - Which mechanisms manage similar dependencies? - Do any mechanisms contribute to specific coordination needs (e.g., need for overview, need to align outputs etc.)? - Any other meaningful way of grouping the mechanisms to use them more efficiently?

5 Discussion

In this section, I will first discuss the implications of my thesis for theory and research on coordination in large-scale agile. First, I discuss how FALC contributes to advancing research on the topic. Second, I discuss theoretical implications and opportunities the framework brings. Third, I discuss the concept of coordination mechanisms as I have come to understand it. Following this, I present the practical implications of this thesis before I outline ideas for future work.

5.1 A comprehensive framework for a maturing field

Previous studies on large-scale agile have identified and described coordination mechanisms from various perspectives. For example, Dingsøy and colleagues report 14 coordination mechanisms used in one of the largest development programs in Norway (Dingsøy, Moe, et al., 2017). In another study, 19 mechanisms were identified (Dingsøy et al., 2018). A longitudinal study reported on how 27 coordination mechanisms were reduced to 14 as the development program transitioned to more mature large-scale development methods (Dingsøy et al., 2022). Moe et al. (2018) reported 15 coordination meetings, and Nyrud and Stray (2017) study identified eleven inter-team mechanisms. All these studies relied on Van de Ven et al.'s (1976) conceptualization. Further, Stray and colleagues used the theoretical frameworks by Strode (2012, 2016) in a study of DevOps teams where they identified 34 coordination mechanisms (Stray, Moe, & Aasheim, 2019). Vedal et al. (2021) reported 22 coordination mechanisms. Another study, where RCT (Gittell, 2006) was used as the theoretical lens, reported seven coordination meetings (Stray, Moe, Vedal, et al., 2022). Finally, in a literature review of 42 case studies, Gustavsson (2017) identified ten roles responsible for inter-team coordination.

The studies exemplified above (and by papers 1 and 2 in this thesis) demonstrate that existing theories from other research fields can be used to analyze how coordination mechanisms manage dependencies across teams. However, a structured approach for collecting, categorizing, and analyzing coordination mechanisms has been lacking until now. Research on large-scale agile software development and coordination is maturing (Palopak et al., 2023), and software engineering researchers have been encouraged to move from applying theories from other fields to generating field-specific theoretical frameworks (Ralph, 2018; Sjøberg et al., 2008; Stol & Fitzgerald, 2015). To this end, FALC, with its specific components, offers the following three contributions to research on coordination in large-scale agile.

First, the taxonomy of inter-team coordination mechanisms can provide a structure for future empirical studies on coordination in large-scale agile. This thesis contributes to existing knowledge of which coordination mechanisms are used in large-scale agile through the 47 reported mechanisms. This collection of mechanisms can be useful for future reference and comparison as researchers continue to describe and report on coordination mechanisms.

Second, the TOPS framework offers a means for analyzing the technical, organizational, physical, and social characteristics of coordination mechanisms. By studying characteristics, researchers may gain a deeper understanding of what it is about a mechanism that makes it manage dependencies well or not in a given large-scale setting (Stray, Moe, Strode, et al., 2022; Strode, 2016; Vedal et al., 2021).

Third, FALC enables researchers to study changes in coordination mechanisms over time. This can further advance knowledge of changes in coordination, which has been called for in previous studies (Dingsøyr et al., 2022; Jarzabkowski et al., 2012; Moe et al., 2018). By analyzing change in coordination mechanisms both in relation to external and internal change events and changes that occur through continuous improvement, researchers can identify which mechanisms are suitable to deal with changing coordination needs in complex large-scale environments.

Importantly, FALC does not make any prescriptive statements of which specific mechanism to use or provide any normative evaluations of whether coordination mechanisms are better or more efficient than others. According to Strode, mechanisms can manage dependencies well, bad, or not at all, and previous research attempts at identifying which mechanisms work better than others have been made (Stray, Moe, Strode, et al., 2022; Vedal et al., 2021). Indeed, some mechanisms will be more efficient in some contexts than others, and some mechanisms are likely not very efficient in large-scale agile at all. FALC, however, is meant to be adaptable to any large-scale context, and therefore, no normative judgments are made. Instead, FALC is intended as a flexible, adaptable tool that researchers may use in different types of large-scale development cases to evaluate and understand what works in that specific setting.

5.2 Towards a theory of coordination mechanisms in large-scale agile

As argued in the section above, much research on coordination mechanisms in large-scale agile has been largely descriptive (Stol & Fitzgerald, 2015). FALC provides the tools for identifying and describing coordination mechanisms in a large-scale context. As research on large-scale agile is maturing (Palopak et al., 2023), the next step is to move towards developing theoretical frameworks that advance our understanding of how coordination mechanisms can be used more effectively to manage dependencies in large-scale agile. This can be done either by building on existing theories or by developing *middle-range theories* that seek to explain the topic in context (i.e., coordination in the context of large-scale agile) (Stol & Fitzgerald, 2015). In this thesis, I have developed an elaborate approach to analyzing coordination mechanisms that form the basis of a middle-range theory of coordination mechanisms in large-scale agile software development.

Gregor (2006) presented a taxonomy of theory types used in research in the information systems field, consisting of *theories for analysis*, *theories for explanation*, *theories for prediction*, and theories that combine explanation and prediction. This taxonomy has previously been used in software engineering research that addresses theory development in the field (e.g., Hannay et al., 2007; Sjøberg et al., 2008; Stol & Fitzgerald, 2015, 2018; Wieringa & Daneva, 2015).

Table 13. Mapping FALC against the components of a theory for explaining (Gregor, 2006)

Overview: FALC is an empirically based theoretical model for analyzing coordination mechanisms in large-scale agile. The model consists of several components, including a taxonomy of coordination mechanisms categories, a framework for coordination mechanisms characteristics, and a model for understanding change in coordination mechanisms.

Component	Instantiation/representation
Means of representation	Words, figures, tables
Primary constructs	Categories of coordination mechanisms: Meetings, roles, and tools and artifacts. Characteristics of coordination mechanisms: Technical, organizational, physical, and social. Change elements: Change events and continuous changes.
Statements of relationship	Examples: -Coordination mechanisms can be grouped in a taxonomy. -Mechanisms have TOPS characteristics.
Scope	Intended for multi-team, large-scale agile software development settings.
Causal explanations	The statements for relationships include causal explanations, for example: -Use of coordination mechanisms enable dependency management. -External change events lead to changes in coordination mechanisms. -Coordination mechanisms can evolve over time because of continuous improvement.
Testable propositions	None.
Prescriptive statements	Partly. Prescriptions for usage of the framework are present but are meant to generate context-specific understandings of each case. There are no normative statements or prescriptions of which mechanisms to use.

Within this taxonomy, FALC constitutes the basis for a theory for explaining. A theory for explaining seeks to address how and why, but it does not aim at formulating testable predictions in a normative sense. Instead, it seeks to show how, when, where, and why events occur, and can therefore also be labeled *theory for understanding* (Gregor, 2006). These types of theories can be either high-level (like structuration theory or actor-network theory) or they can be at a lower level, including middle-range theories (Stol & Fitzgerald, 2015).

In Table 10, I use the components of a theory for explaining as described by Gregor (2006) to outline the beginning of such a theory which is to be further developed in future work.

5.3 An actionable conceptualization of coordination mechanisms in large-scale agile.

There is no unified definition of coordination or coordination mechanisms across research fields. In Section 2.2, I reviewed some influential approaches to the study of coordination and their different views on coordination mechanisms. In Section 4.1, I introduced the taxonomy of inter-team coordination mechanisms, consisting of scheduled and unscheduled meetings, individual and team roles, and tangible and intangible tools and artifacts.

The 47 coordination mechanisms identified at Entur can all be categorized according to the taxonomy. However, using a different conceptualization of coordination and coordination mechanisms, it is possible to argue that other elements can also constitute coordination

mechanisms. For example, in Paper 2, we included *open office space* and *co-location* as coordination mechanisms because we applied Strode's (2012) theory of coordination. Since then, new insights have led me to no longer consider these as coordination mechanisms, and they were excluded from the list of mechanisms presented in Section 4.1 in tables 8-10. Based on my current understanding of coordination mechanisms, I now argue that co-location and open office spaces enable coordination but do not in themselves manage dependencies. Being co-located in an open office space enables coordination by providing opportunities for physical meetings and using tangible tools and artifacts. Co-location also enables swift and timely ad hoc coordination, which has been shown both from my data and from other studies (e.g., Dingsøyr et al., 2022; Hoda & Noble, 2017; Hussain et al., 2022; Strode et al., 2012). However, many coordination mechanisms can be used regardless of whether the space is co-located or distributed. Indeed, much inter-team coordination happens digitally also in co-located settings by using coordination tools, such as Slack (Moe et al., 2018; Stray, Moe, Vedal, et al., 2022). At Entur, coordination across teams would often happen on Slack even though people were co-located in the same office space. These notions have become even more relevant today when coordination happens on a spectrum of hybrid work arrangements (Smite et al., 2022; Sporseem et al., 2022).

In a related vein, other abstract elements considered as coordination mechanisms include, for example, rules and representations (Okhuysen & Bechky, 2009), standardization (Malone & Crowston, 1994), and convention (Schmidt & Simonee, 1996). Throughout this Ph.D. project, I have come to understand coordination mechanisms as more concrete entities that can be made available for use in a physical or digital space. By limiting the coordination mechanism categories to concrete elements, I believe it may be easier to understand how a mechanism manages dependencies by allowing for concrete reasoning around its use. Analyzing a meeting, for example, by considering who is present, the meeting's goals and outcomes, and how this relates to inter-team dependencies can serve to raise awareness of coordination effectiveness. Accordingly, by presenting the taxonomy of inter-team coordination and FALC, I aim to provide a more actionable approach to coordination mechanisms that may be easier to apply in practice. Furthermore, I offer the following definition of inter-team coordination mechanisms:

A coordination mechanism is an organizational process, entity, or arrangement used to manage dependencies to realize a collective performance. When used to manage dependencies between two or more teams, this can be defined as an inter-team coordination mechanism.

This definition is influenced by the work of others, in particular by Malone and Crowston's (1994) basic definition of coordination, combined with Okhuysen and Bechky (2009)'s view of coordination mechanisms as processes, roles, routines, and arrangements. It also recognizes the importance of artifacts and similar entities (Schmidt & Simonee, 1996). As research on inter-team coordination continues to mature, a clear and specific definition contributes to advancing knowledge and future studies on inter-team coordination mechanisms in large-scale agile.

5.4. Implications for practice

The findings and contributions in this thesis lead to several practical implications.

First, my analyses have shown that inter-team meetings are important for managing dependencies. At Entur, 21 different meetings contributed to inter-team coordination. Of these, 15 were inter-team meetings. However, not all meetings are equally effective across organizational contexts and across time. A specific example at Entur was the product owner meetings, which were adapted over time as their effectiveness at managing and prioritizing dependencies changed. Being mindful of which meetings are used and for what appears essential as much time is spent in meetings which, in turn, have high costs (Moe et al., 2018; Stray et al., 2018; Stray & Moe, 2020).

Second, it is important to consider which roles are used to manage dependencies at an inter-team level and how this is done. At Entur, there were 13 coordination roles, eight of which were located at the inter-team level. Three roles, the product owners, team leaders, and tech leads, were considered as part of the development teams but still performed much inter-team coordination. In other organizations, these roles may be conceptualized differently, such as the product owner role, which is often understood as an inter-team role (Bass & Haxby, 2019; Paasivaara et al., 2012). Using temporary task force teams was identified as a success factor at Entur. Therefore, the use of such temporary teams with the related temporary team arrangements is recommended in large-scale development programs that have shorter-term projects with inter-team implications.

Third, gaining an overview of tool and artifact usage provides another opportunity for improving dependency management. Like Entur, many organizations may find that they use several communication tools but that one or a few are dominating. As demonstrated in this thesis, at Entur, Slack was a powerful communication tool that enabled efficient inter-team coordination. However, over time, the need to adapt as the program and its client base grew led to changes in how the mechanisms were used. Spending some time reflecting on optimal tool use can be worth the time and money, as research has found that developers use much time on Slack and similar tools and are often interrupted in their workflow by new messages, but also that these tools provide a means for collegial socializing and support (Giuffrida & Dittrich, 2015; Stray, Moe, & Noroozi, 2019; Stray & Moe, 2020). Understanding how to best facilitate coordination via communication tools is therefore important, in particular because many, if not most, of these programs (like Slack and Microsoft Teams) often have high usage fees (Lin et al., 2016; Stray & Moe, 2020).

These and other examples demonstrate that, from a practical perspective, being aware of coordination mechanism use is important. However, as all contexts differ, I make no specific recommendation on which mechanisms to use. Instead, I encourage to be mindful of their unique coordination setting and adapt their practices accordingly. Practitioners can apply FALC to understand and improve their current coordination situation. The approach is context-sensitive and non-normative in that it does not prescribe what to do, which mechanisms to use, or how to use them. Instead, the model provides a thinking tool that may be readily applied by practitioners seeking to understand their coordination situation and use the knowledge they derive to make their own decisions. The instructions provided in Section 4.3 are designed to be ready for use and experimentation from a practical perspective.

5.5 Future work

This thesis contributes to the body of empirical studies on coordination in large-scale agile and gives rise to opportunities for future work. Using ethnographic methods, I have documented in detail the data collected over 1.5 years of fieldwork. I have also rigorously documented the case organization and the analytical procedures. These descriptions contribute to the usefulness and trustworthiness of the findings and conclusions drawn from this research and provide opportunities for following up with new empirical studies. In particular, FALC can be used for future studies on coordination in large-scale agile, both empirical and conceptual.

First, FALC is derived from empirical work, but it has yet to be used in an empirical setting. Therefore, one specific avenue for future work is to use and test the framework in new large-scale settings, preferably both in co-located and distributed settings. Moreover, the framework's components should be scrutinized for further development and refinement. In Paper 3, we recommended that the taxonomy of inter-team coordination should be evaluated against existing quality criteria, and we proposed using the criteria outlined by Nickerson et al. (2013). These or other quality assurance criteria should be applied to the framework as a whole, including the taxonomy, the TOPS framework, and the model of change in coordination mechanisms. Table 11 shows how FALC currently performs across the six dimensions proposed by Nickerson et al. (2013) and outlines opportunities for future work.

Second, in this analysis I kept the mapping of dependencies at a high level, in that I reported on which coordination meetings, roles, and tools and artifacts managed which type of dependencies in terms of knowledge, process, and resource dependencies. It would be interesting to go into a more detailed level of analysis on the types of dependencies managed, by applying the full scale of Strode's (2016) dependency taxonomy.

Third, FALC could assist future conceptual and review studies of the coordination literature. The framework allows for identifying and categorizing all mechanisms used for dependency management across large-scale contexts and is not limited to the 47 mechanisms identified at Entur. New mechanisms, such as the product manager role (Tkalich et al., 2022) and the OKR tracker (Stray, Moe, Vedal, et al., 2022), are generated as software development practices evolve and mature. These can easily be categorized in the taxonomy, which demonstrates the extendibility and explanatory ability of the taxonomy. It would be interesting to see a full review of the literature on coordination mechanisms using FALC, which would also contribute to improving and refining the framework.

Fourth, as described in Section 2.2.1 and Table 3, there are several definitions of large-scale agile available, all focusing on different aspects and characteristics of large-scale agile. Future work should aim to develop a clear and general definition of the term 'large-scale agile' that is sensitive to size, development practices, and the need to coordinate across teams. The descriptions of the large-scale agile environment in this dissertation can serve as input to this work.

Table 14. Evaluating FALC against the quality criteria by Nickerson et al. (2013).

Criterion	Description	How the criterion is met in this thesis
Conciseness	A taxonomy or framework should be parsimonious, containing a limited number of dimensions and categories.	Because FALC contains a limited number of dimensions, i.e., the three taxonomy categories with a total of six sub-categories, the four TOPS characteristics, and three dimensions of the change model, this criterion is met.
Robustness	The categories should be sufficient in number and content to clearly differentiate the objects included. They should further be mutually exclusive and disallow overlap.	The included categories appear sufficient to capture all inter-team coordination mechanisms observed from our data. The categories are further mutually exclusive, i.e., a meeting is sufficiently different from a tool, a technical characteristic is different from a physical characteristic. Internal and external change events are clearly distinct, however future research can further refine continuous versus internal changes.
Comprehensiveness	In sum, the categories should be able to include all known objects in the domain.	While the categories cover all objects observed in the empirical case, it is possible that future research will discover additional categories. More research using the framework is needed to meet this criterion.
Extendibility	The taxonomy should not be fixed, but open to extensions as applicable, meaning that it should be possible to add new categories if new types of objects occur.	Should new categories be needed based on other empirical observations or studies there is room to add these as applicable. The framework is thus extendible.
Explanatory ability	The categories should be at a level of abstraction that allows for the inclusion of new objects, and precise enough to allow for easy identification of an object's placement in the categories.	The analysis carried out for this thesis led to the inclusion of ten new mechanisms, which could readily be placed in the taxonomy and categorized using TOPS. However, other researchers should also apply these elements to test if the explanatory ability holds.
Usability*	The taxonomy is useful if it is used by others over time.	The final criterion is met if, over time, FALC is used in other studies and by other researchers within the domain.

*This criterion is not part of Nickerson et al.'s (2013) list but was included by Strode (2016).

5.6 Reflections on the research process

In this final section of the discussion chapter, I take a step back and provide some reflections on the research process, my role as a researcher, the ethical aspects of my research as well as the research limitations.

5.6.1 The role of the researcher

Even though the interpretive paradigm acknowledges social construction as part of the research process, the interpretive researcher needs to be mindful of the role of the researcher in shaping the research process from start to end. Our background and previous experiences, our areas of research interest, level of competence, and social skills, among others, influence what we notice, what we consider relevant, and what we will both consciously and unconsciously ignore

(Crang & Cook, 2007; Walsham, 2002, 2012). In the following, I will draw on some of my personal experiences to illustrate some of these points.

I currently consider myself an interpretive software engineering researcher. However, my academic upbringing is positivist, influenced by the fields of economics, organization science, and psychology. Further, the research I conducted before starting the Ph.D. was quantitative. These past experiences have undoubtedly influenced the research process. I struggled for a while to come to terms with the social constructivist nature of my qualitative research, and I have more than once doubted whether all my observational notes, analytical memos, and interview transcripts would be rigorous enough in the end. However, I chose to trust the methods I had selected, and today I see the large body of textual material I've collected as a great asset.

Further, our involvement style as researchers influence how we interpret what we see, the information we are given, and how we engage with participants (Crang & Cook, 2007; Walsham, 2006). Before returning to academia, I worked with HR and recruitment in the IT consulting industry. The way I engaged with people on-site, as well as how I chose to conduct my interviews, has been influenced by the many professional job interviews I have conducted in the past, as well as the “friendly HR person” way of engaging with people, for better or worse. I believe I was successful in establishing rapport with most participants. During interviews, I am accustomed to taking a listening and open stance, and overall, I have the impression that people felt it was safe to talk to me. At the same time, this style of involvement may have led me to miss out on chances to “dig deeper” out of politeness on certain occasions. Over time, the involvement style may change as the field researcher becomes more familiar with and to the research context. I noticed this challenge more toward the end of the data collection. When I knew the case and many of the people working there well and felt safe in the setting and my role, it was easier to ask more inquisitive questions than at the beginning of the fieldwork.

Finally, my background in organizational psychology influenced the extent to which I have focused on the actual software under development and the role of technical aspects shaping the coordination process during the data collection. However, software engineering as a research field is heavily influenced by the social sciences, and human aspects have been a core part of software engineering research since the early 2000s (Hoda et al., 2018). Nevertheless, had my background been in computer science, I would probably have noticed and focused on slightly different aspects during fieldwork and analysis of the data. During this Ph.D. project, I have learned much about software engineering and computer science. If I were to conduct the study again, I would have liked to dig deeper into areas like architecture and study technical dependencies in more detail.

Such issues are not necessarily to be avoided within the interpretive paradigm (Klein & Myers, 1999; Walsham, 2002, 2012). However, it is important to recognize them and be clear and upfront about their consequences for the research (Crang & Cook, 2007; Edwards & Holland, 2013; Walsham, 2006). For example, with interviews, the researcher should ideally keep an open, listening, and non-directive stance and not impose opinions on the participant or influence their responses (Crang & Cook, 2007; Edwards & Holland, 2013). In reality, an interview is an intersubjective event where both researcher and participant co-create the

interview through interactional dynamics (Blomberg et al., 1993; Edwards & Holland, 2013; Stake, 2005), no matter how open the interviewer tries to be. Similarly, during observation, I did not take the position of the disengaged observer detached from the social scene. I engaged with people, was asked for my opinion and advice, and helped with smaller tasks. I recognize the intersubjectivity developed between the participants and myself as I gradually became more immersed in the observation context.

5.6.2 Ethical considerations

There are always ethical considerations when conducting research that involves humans, no matter type or form (Crang & Cook, 2007). This includes both practical considerations, as well as more subtle challenges, and there are many grey areas (Walsham, 2012). A primary ethical challenge is participant anonymity in the data, which contains descriptive information about events and people. This project has been approved by and adheres to the Norwegian Agency for Shared Services in Education and Research standards, such as anonymizing participants and storing data safely. I sent out information letters and collected consent forms before tape-recording interviews or collecting any other explicitly personal data (See Appendix A). However, there are also more subtle considerations and grey areas associated with research ethics (Walsham, 2012). Two examples are ensuring participants' actual and ongoing consent to being observed and disclosing and disseminating findings to the participants. To address the former, I made all employees were informed about my presence at an overall level, and I never took photos without asking people. But I did not obtain consent forms for each meeting observed, lunches, or other social interaction, as it was considered too intrusive and time-consuming by the case representatives. For the latter example, I sent each new paper to case representatives for review, and I also offered these to interview participants. I have also presented findings at company-wide events.

Beyond such formal procedures, researcher sensitivity is critical in case study research, in particular when fieldwork is involved, compared to for instance, survey research where the researcher is more distanced and detached (Walsham, 2012). During my fieldwork, people sometimes told me things I did not take further, and it is important to safeguard this type of information in a way that protects the participants (Crang & Cook, 2007; Stake, 2005; Walsham, 2012). Here, judgment calls will often have to be made. Referring to the above discussion of whether my research could have fallen under the critical paradigm, if I were to go into the power structures at play, I would have been likely to encounter more controversial issues that would have required even more discretion on my part.

5.6.3 Limitations

All research has methodological and analytical limitations. These limitations vary with the type of study (Yin, 2018). One limitation of this thesis is that I have not been able to account for all that has been written on coordination. As described in Chapter 2, coordination is a widely interdisciplinary field of study, with several existing theories to explain how coordination occurs. In Chapter 2, I presented some of the theories and research approaches to coordination, focusing on those I have found to be most relevant and influential for research on large-scale agile. Despite this conscious and reflected choice of scope, I acknowledge that I may have

missed out on relevant contributions. Ideally, one should always be able to conduct systematic literature reviews (SLR) as part of any new research project. However, SLRs are time consuming, and represent research contributions in themselves. As such, there is a trade-off in terms of what can be achieved within the scope of a Ph.D. project. For this thesis, the choice was made to focus on obtaining rich, high-quality empirical data, admittedly at the expense of a full account for all theoretical approaches to, and all relevant conducted on, research coordination.

Another limitation of this Ph.D. thesis is that it is based on a single case, where the qualitative data were, by and large, collected and administered by one person. This raises questions about the trustworthiness and usability of the findings (Yin, 2018). Because the analyses are interpretive thematic analyses, conducted from an interpretivist epistemological point of view, the hypothetico-deductive logic of assessing reliability and different forms of validity is less relevant (Nowell et al., 2017; Stake, 2005; Yazan, 2015). It is nevertheless important to assess the quality of the research.

In Section 3.1, I reflected upon the choice of a single case. I argued that despite the limitations associated with single-case research, the choice is warranted as I consider Entur a “critical case” for coordination in large-scale agile. Such cases can be evaluated using the quality criteria for assessing the trustworthiness of naturalistic inquiries (Guba, 1981; Lincoln & Guba, 1985), which consist of *credibility*, *transferability*, *dependability*, and *confirmability*. These criteria can be applied both within positivist and interpretive paradigms, and they are also used within the software engineering field (Hussain et al., 2022; Molléri et al., 2023; Russo, 2021). Nowell et al. (2017) provide guidance on how to meet these criteria when conducting thematic analysis. In the following, I comment upon each of these terms.

First, *credibility* refers to the “fit” between the views and representations of the research participants and the researcher’s representation of them (Guba, 1981; Nowell et al., 2017). When the descriptions of a study appear sound and recognizable, they can be said to be credible (Lincoln & Guba, 1985). There are several methods and techniques for establishing credibility, such as long-term engagement with the research site or participants, ongoing and persistent observation, researcher, and data collection triangulation, and conducting external checks and participant member checks (Nowell et al., 2017). In the seminal work on thematic analysis, the much-used term “triangulation” is not used (Braun & Clarke, 2006, 2012). However, the use of several data sources to collect the data (i.e., data triangulation) and having several researchers engaged in the interpretation of the analysis (i.e., researcher triangulation) is encouraged and part of a good analytical process (Braun & Clarke, 2012, 2021a). In the research included in this thesis, I used all these techniques. The long-term fieldwork and the various data sources, the use of supervisors and other co-authors throughout the research project and having both Entur employees as well as colleagues and reviewers assessing the papers, all contributed to the credibility criterion.

Second, *transferability* relates to the generalizability of the findings. This does not mean absolute external generalizability in a positivist sense, rather it refers to theoretical generalizability made possible through thick descriptions of the research process and the findings (Nowell et al., 2017). When this criterion is met, readers can understand the research reported and judge the generalizability to their own settings based on the descriptions made. In

the case of this research, the transferability criterion is addressed through the detailed descriptions of the research process, the case, and the findings provided throughout the research. Further, as Entur represents a critical case for large-scale agile (Flyvbjerg, 2006), other large-scale agile organizations should be able to recognize the settings described.

Third, *dependability* is achieved when the research process is traceable, logical, and clearly documented. If readers are able to examine the research process, they can better judge its dependability (Nowell et al., 2017). I have been careful in trying to describe as much detail as possible on the data collection, analytical procedures, and findings of my research, both throughout this thesis summary, as well as in the four research papers. Nevertheless, it is rarely possible to examine the entire research process, as this would be very time-consuming and often also impossible due to underlying anonymity clauses and ethics clearances, which is the case also with this research. However, having the research process audited is one way for a study to demonstrate dependability (Nowell et al., 2017). Naturally, as a Ph.D. project, this research was supervised by two experienced researchers who followed the process closely.

Finally, *confirmability* refers not to the extent to which others are able to replicate the exact same findings, which is not a goal of interpretive research. Rather, confirmability, in this case, refers to the extent to which the interpretations and findings are clearly described such that others can understand how the conclusions have been made (Nowell et al., 2017). I hope that the detailed descriptions in Chapter 3 and Chapter 4, as well as the research papers, will allow the reader to understand what has been done and why. Notwithstanding this, there are some weaknesses both in the reporting and the research itself. For example, as described in sections 4.1 and 4.2, I applied the taxonomy of dependencies (Strode, 2016) as part of the analyses for RQ1 and RQ2, but only at a high level, reporting on the three main dependency categories rather than going into detail applying the sub-categories. I believe this approach may have been too superficial and that a more fine-grained application of the dependency taxonomy could have led to a more nuanced understanding of which coordination mechanisms manage which dependencies, and how. This choice was motivated by the goal of presenting the novel findings of my own research rather than demonstrating the confirmability of existing research. I nevertheless acknowledge this as a limitation that can affect the transferability and confirmability of the research, as the high-level reporting can make it more difficult for others to understand the findings.

Furthermore, the novel framework, FALC, needs empirical testing. In Chapter 5, I evaluate FALC on a theoretical level by mapping it against the criteria for a theory of explaining and against the quality criteria for evaluating research taxonomies and frameworks (see tables 13 and 14), but further testing using actual data is needed. As the framework has been developed from single-case data, it is necessary to test how the framework transfers to other settings, which is likely to produce new insights and further shape the framework and how it is to be used. As mentioned in Section 5.5., this constitutes an arena for future research. FALC's usability can be further determined only by the extent to which it is used in future research and/or practice. It is my hope that the detailed descriptions provided throughout the papers and this thesis summary, as well as the thorough descriptions of the theoretical, methodological, and analytical choices made throughout the research process, contribute to the confirmability of the research.

6 Conclusion

Managing dependencies through various coordination mechanisms is crucial for successful large-scale agile software development. Previous research has focused on identifying coordination mechanisms and describing how they support coordination, often focusing on changes in coordination following some event, such as a large-scale agile transformation or the implementation of a large-scale agile framework. However, a structured approach to analyzing coordination mechanisms, categorizing them and their characteristics, to understand how they support dependencies and how they change over time has been lacking.

In the introduction chapter, the thesis statement for this dissertation was presented as follows: *“In large-scale agile software development, dependencies within and between teams must be managed by using coordination mechanisms. A better understanding of which coordination mechanisms can be used to manage dependencies and how these mechanisms may change in response to a complex large-scale environment will contribute to improving coordination, thereby facilitating more successful software development and continuous value delivery.”* Based on this statement, the objective of this Ph.D. thesis was *“to collect evidence about which coordination mechanisms are used, in what way, to manage dependencies, and to synthesize this evidence to advance knowledge on coordination in large-scale agile”*.

To fulfill this objective, I conducted a longitudinal field study using ethnographic methods, following a large-scale agile development program for 1.5 years and collecting a rich and varied data material. The data was analyzed several times through thematic analysis, which resulted in four papers and a thesis summary that has provided a detailed and nuanced understanding of how coordination mechanisms in large-scale agile.

This thesis summary addressed three research questions. The first research question was, *“Which coordination mechanisms are used to manage inter-team dependencies in large-scale agile software development?”* In response to this question, I have reported 47 unique coordination mechanisms that were used in the case organization. Re-analyzing the coordination mechanisms also led to three new insights, that is, 1) that team-level mechanisms can manage inter-team dependencies, 2) that some mechanisms should be collapsed into overall types, while others should be split into unique mechanisms, and 3) that some elements should be excluded compared to the original papers.

The second research question was, *“How are coordination mechanisms used to manage dependencies in large-scale agile software development?”* In Section 4.2, I provided three detailed examples of how specific mechanisms were used at Entur to manage dependencies, namely the tech lead forum, the product owner role, and the communication tool Slack. These examples were based on the papers, which all address how coordination mechanisms are used to support dependency management across teams from different angles.

The third research question, *“How can we analyze coordination mechanisms in large-scale agile software development?”* was answered by introducing FALC, the Framework for Analyzing Large-scale agile Coordination mechanisms. FALC is an empirically based theoretical framework and consists of three main components: 1) A taxonomy of inter-team coordination mechanisms which can provide structure for future empirical and analytical work

on the research topic; 2) a framework for analyzing the technical, organizational, physical, and social (TOPS) characteristics of coordination mechanisms, which allows for a deeper understanding of how individual mechanisms contributes to managing dependencies and 3) a model to study changes in coordination mechanisms over time.

FALC provides tools for structuring and organizing empirical studies on coordination mechanisms. By using FALC in new empirical studies, future research can also contribute to further developing the framework. Practitioners can apply the framework as input to an analysis to understand and improve their current coordination situation. The framework is context-sensitive and non-normative in that it does not prescribe what to do or which mechanisms to use, or how to use them. Instead, the framework provides a thinking tool that can be readily applied by practitioners seeking to understand their coordination situation and use the knowledge they derive to make their own decisions.

By this, the Ph.D. thesis advances knowledge on coordination mechanisms in large-scale agile, with the goal of assisting future research and practice in improving dependency management in large-scale agile software development. The knowledge derived is hoped to contribute to successful software development and continuous value delivery, ultimately providing value to both the software customers and society as a whole.

References

- Ågren, P., Knoph, E., & Berntsson Svensson, R. (2022). Agile software development one year into the COVID-19 pandemic. *Empirical Software Engineering*, 27(6), 1–50.
- Andriyani, Y., Hoda, R., & Amor, R. (2017). Reflection in agile retrospectives. 3–19.
- Bass, J. M. (2015a). How product owner teams scale agile methods to large distributed enterprises. *Empirical Software Engineering*, 20(6), 1525–1557.
- Bass, J. M. (2016). Artefacts and agile method tailoring in large-scale offshore software development programmes. *Information and Software Technology*, 75, 1–16.
- Bass, J. M., & Haxby, A. (2019). Tailoring product ownership in large-scale agile projects: Managing scale, distance, and governance. *IEEE Software*, 36(2), 58–63.
- Bass, J. M., & Salameh, A. (2020). Agile at scale: A summary of the 8th International Workshop on Large-Scale Agile Development. 68.
- Batra, D., Xia, W., VanderMeer, D. E., & Dutta, K. (2010). Balancing agile and structured development approaches to successfully manage large distributed software projects: A case study from the cruise line industry. *CAIS*, 27, 21.
- Berntzen, M., Hoda, R., Moe, N. B., & Stray, V. (2022). A taxonomy of inter-team coordination mechanisms in large-scale agile. *IEEE Transactions on Software Engineering*, 49(2), 699–718. <https://doi.org/doi: 10.1109/TSE.2022.3160873>
- Berntzen, M., Moe, N. B., & Stray, V. (2019). The Product Owner in Large-Scale Agile: An Empirical Study Through the Lens of Relational Coordination Theory. In P. Kruchten, S. Fraser, & F. Coallier (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (pp. 121–136). Springer International Publishing.
- Berntzen, M., Stray, V., & Moe, N. B. (2021). Coordination Strategies: Managing Inter-team Coordination Challenges in Large-Scale Agile. In P. Gregory, C. Lassenius, X. Wang, & P. Kruchten (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (pp. 140–156). Springer International Publishing.
- Berntzen, M., Stray, V., Moe, N. B., & Hoda, R. (in press). Responding to Change Over Time: A Longitudinal Case Study on Changes in Coordination Mechanisms in Large-Scale Agile. *Empirical Software Engineering*.
- Berntzen, M., & Wong, S. I. (2019). Coordination in Distributed, Self-managing Work Teams: The Roles of Initiated and Received Task Interdependence. *Proceedings of the 52nd Hawaii International Conference on System Sciences*.
- Berntzen, M., & Wong, S. I. (2021). Autonomous but Interdependent: The Roles of Initiated and Received Task Interdependence in Distributed Team Coordination. *International Journal of Electronic Commerce*, 25(1).
- Bick, S., Spohrer, K., Hoda, R., Scheerer, A., & Heinzl, A. (2018). Coordination challenges in large-scale software development: A case study of planning misalignment in hybrid settings. *IEEE Transactions on Software Engineering*, 44(10), 932–950.
- Blomberg, J., Giacomi, J., Mosher, A., & Swenton-Wall, P. (1993). Ethnographic field methods and their relation to design. In D. Schuler & A. Namioka (Eds.), *Participatory design: Principles and practices* (Vol. 7, pp. 123–155). Lawrence Erlbaum Assoc.

- Bolton, R., Logan, C., & Gittel, J. H. (2021). Revisiting relational coordination: A systematic review. *The Journal of Applied Behavioral Science*, 57(3), 290–322.
- Bourdieu, P. (1977). *Outline of a Theory of Practice* (Issue 16). Cambridge university press.
- Bozan, K. (2017). The Perceived Level of Collaborative Work Environment's Effect on Creative Group Problem Solving in a Virtual and Distributed Team Environment. *Proceedings of the 50th Hawaii International Conference on System Sciences*.
- Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2), 77–101.
- Braun, V., & Clarke, V. (2012). Thematic analysis. In *APA handbook of research methods in psychology, Vol 2: Research designs: Quantitative, qualitative, neuropsychological, and biological*. (pp. 57–71). American Psychological Association.
- Braun, V., & Clarke, V. (2019). Reflecting on reflexive thematic analysis. *Qualitative Research in Sport, Exercise and Health*, 11(4), 589–597.
- Braun, V., & Clarke, V. (2021a). One size fits all? What counts as quality practice in (reflexive) thematic analysis? *Qualitative Research in Psychology*, 18(3), 328–352.
- Braun, V., & Clarke, V. (2021b). To saturate or not to saturate? Questioning data saturation as a useful concept for thematic analysis and sample-size rationales. *Qualitative Research in Sport, Exercise and Health*, 13(2), 201–216.
- Braun, V., Clarke, V., & Hayfield, N. (2022). 'A starting point for your journey, not a map': Nikki Hayfield in conversation with Virginia Braun and Victoria Clarke about thematic analysis. *Qualitative Research in Psychology*, 19(2), 424–445.
- Campanelli, A. S., & Parreiras, F. S. (2015). Agile methods tailoring—A systematic literature review. *Journal of Systems and Software*, 110, 85–100.
- Carroll, N., Conboy, K., & Wang, X. (2023). From Transformation to Normalisation: An Exploratory Study of a Large-Scale Agile Transformation. *Journal of Information Technology*, 02683962231164428. <https://doi.org/10.1177/02683962231164428>
- Carstensen, P. H., & Sørensen, C. (1996). From the social to the systematic. *Computer Supported Cooperative Work (CSCW)*, 5(4), 387–413.
- Cataldo, M., & Herbsleb, J. D. (2012). Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering*, 39(3), 343–360.
- Claggett, J. L., & Karahanna, E. (2018). Unpacking the structure of coordination mechanisms and the role of relational coordination in an era of digitally mediated work processes. *Academy of Management Review*, 43(4), 704–722.
- Conboy, K., & Carroll, N. (2019). Implementing large-scale agile frameworks: Challenges and recommendations. *IEEE Software*, 36(2), 44–50.
- Conway, M. E. (1968). How do committees invent. *Datamation*, 14(4), 28–31.
- Crang, M., & Cook, I. (2007). *Doing ethnographies*. Sage.
- Crowston, K., & Osborn, C. S. (2003). A coordination theory approach to process description and redesign. In T. W. Malone, K. Crowston, & G. A. Herman (Eds.), *Organizing Business Knowledge: The MIT Process Handbook* (pp. 335–370). MIT Press.
- Digital.ai. (2020). 14th Annual State of Agile Report (2019) (Annual State of Agile Report). <https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494>

- Digital.ai. (2021). 15th Annual State of Agile Report (2020). <https://digital.ai/resource-center/analyst-reports/state-of-agile-report>
- Digital.ai. (2022). 16th State of Agile Report. <https://digital.ai/resource-center/analyst-reports/state-of-agile-report/>
- Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, 119, 87–108.
- Dingsøy, T., Bjørnson, F. O., Schrof, J., & Sporse, T. (2022). A longitudinal explanatory case study of coordination in a very large development programme: The impact of transitioning from a first- to a second-generation large-scale agile development method. *Empirical Software Engineering*, 28(1), 1. <https://doi.org/10.1007/s10664-022-10230-6>
- Dingsøy, T., Fægri, T. E., & Itkonen, J. (2014). What is large in large-scale? A taxonomy of scale for agile software development. In *International Conference on Product-Focused Software Process Improvement* (pp. 273–276). Springer, Cham.
- Dingsøy, T., Falessi, D., & Power, K. (2019). Agile development at scale: The next frontier. *IEEE Software*, 36(2), 30–38.
- Dingsøy, T., Moe, N. B., Fægri, T. E., & Seim, E. A. (2017). Exploring software development at the very large-scale: A revelatory case study and research agenda for agile method adaptation. *Empirical Software Engineering*, 1–31.
- Dingsøy, T., Moe, N. B., & Seim, E. A. (2018). Coordinating Knowledge Work in Multi-Team Programs: Findings from a Large-Scale Agile Development Program. *Project Management Journal*, 49, 64–77.
- Dingsøy, T., Rolland, K., Moe, N. B., & Seim, E. A. (2017). Coordination in multi-team programmes: An investigation of the group mode in large-scale agile software development. *Procedia Computer Science*, 121, 123–128.
- Dittrich, Y. (2016). What does it mean to use a method? Towards a practice theory for software engineering. *Information and Software Technology*, 70, 220–231.
- Dittrich, Y., Nørbjerg, J., Tell, P., & Bendix, L. (2018). Researching cooperation and communication in continuous software engineering. 87–90.
- Easterbrook, S., Singer, J., Storey, M.-A., & Damian, D. (2008). Selecting empirical methods for software engineering research. In F. Shull, J. Singer, & D. I. K. Sjøberg (Eds.), *Guide to advanced empirical software engineering* (pp. 285–311). Springer.
- Edison, H., Wang, X., & Conboy, K. (2022). Comparing Methods for Large-Scale Agile Software Development: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 48(8), 2709–2731. <https://doi.org/10.1109/TSE.2021.3069039>
- Edwards, R., & Holland, J. (2013). *What is qualitative interviewing?* A&C Black.
- Espinosa, J. A., Slaughter, S. A., Kraut, R. E., & Herbsleb, J. D. (2007). Team knowledge and coordination in geographically distributed software development. *Journal of Management Information Systems*, 24(1), 135–169.
- Faraj, S., & Sproull, L. (2000). Coordinating expertise in software development teams. *Management Science*, 46(12), 1554–1568.
- Faraj, S., & Yan, X. (2006). Coordination in Fast-Response Organizations. *Management Science*, 52(8), 1155–1169.

- Flyvbjerg, B. (2006). Five misunderstandings about case-study research. *Qualitative Inquiry*, 12(2), 219–245.
- Fowler, M., & Highsmith, J. (2001). The Agile Manifesto. <http://agilemanifesto.org/>
- Geertz, C. (1973). *The interpretation of cultures* (Vol. 5043). Basic books.
- Giddens, A. (1993). *New rules of sociological method*. Stanford University Press.
- Gittell, J. H. (2002). Coordinating mechanisms in care provider groups: Relational coordination as a mediator and input uncertainty as a moderator of performance effects. *Management Science*, 48(11), 1408–1426.
- Gittell, J. H. (2006). Relational coordination: Coordinating work through relationships of shared goals, shared knowledge and mutual respect. *Relational Perspectives in Organizational Studies: A Research Companion*, 74–94.
- Gittell, J. H. (2012). *New Directions for Relational Coordination Theory*. Oxford University Press. <https://doi.org/10.1093/oxfordhb/9780199734610.013.0030>
- Gittell, J. H., & Douglass, A. (2012). Relational bureaucracy: Structuring reciprocal relationships into roles. *Academy of Management Review*, 37(4), 709–733.
- Giuffrida, R., & Dittrich, Y. (2015). A conceptual framework to study the role of communication through social software for coordination in globally-distributed software teams. *Information and Software Technology*, 63, 11–30.
- Govers, M., & van Amelsvoort, P. (2018). A socio-technical perspective on the design of IT architectures: The lowlands lens. *Management Studies*, 6(3), 177–187.
- Guba, E. G. (1981). Criteria for assessing the trustworthiness of naturalistic inquiries. *Ectj*, 29(2), 75–91.
- Gustavsson, T. (2017). Assigned roles for Inter-team coordination in Large-Scale Agile Development: A literature review. 1–5.
- Gustavsson, T. (2019a). Dynamics of Inter-Team Coordination Routines in Large-Scale Agile Software Development. *Proceedings of the 27th European Conference on Information Systems (ECIS)*, 1–16. DiVA. <http://urn.kb.se/resolve?urn=urn:nbn:se:kau:diva-80166>
- Gustavsson, T., Berntzen, M., & Stray, V. (2022). Changes to team autonomy in large-scale software development: A multiple case study of Scaled Agile Framework (SAFe) implementations. *International Journal of Information Systems and Project Management*, 10(1), 29–46.
- Hannay, J. E., Sjöberg, D. I., & Dyba, T. (2007). A systematic review of theory use in software engineering experiments. *IEEE Transactions on Software Engineering*, 33(2), 87–107.
- Herbsleb, J. D. (2007). Global software engineering: The future of socio-technical coordination. In *Future of Software Engineering (FOSE'07)* (pp. 188–198). IEEE.
- Herbsleb, J. D. (2016). Building a socio-technical theory of coordination: Why and how (outstanding research award). 2–10.
- Herbsleb, J. D., & Mockus, A. (2003). Formulation and preliminary test of an empirical theory of coordination in software engineering. *ACM SIGSOFT Software Engineering Notes*, 28(5), 138–137.
- Hoda, R. (2021). Socio-Technical Grounded Theory for Software Engineering. *IEEE Transactions on Software Engineering*. <https://doi.org/10.1109/TSE.2021.3106280>

- Hoda, R., & Noble, J. (2017). Becoming agile: A grounded theory of agile transitions in practice. 141–151.
- Hoda, R., Noble, J., & Marshall, S. (2012). Self-organizing roles on agile software development teams. *IEEE Transactions on Software Engineering*, 39(3), 422–444.
- Hoda, R., Salleh, N., & Grundy, J. (2018). The rise and evolution of agile software development. *IEEE Software*, 35(5), 58–63.
- Howison, J., Rubleske, J., & Crowston, K. (2015). Coordination Theory: A Ten-Year Retrospective. In *Human-computer Interaction and Management Information Systems: Foundations* (pp. 134–152). Routledge.
- Hukkelberg, I., & Berntzen, M. (2019). Exploring the Challenges of Integrating Data Science Roles in Agile Autonomous Teams. In R. Hoda (Ed.), *Agile Processes in Software Engineering and Extreme Programming – Workshops* (pp. 37–45). Springer International Publishing.
- Hussain, W., Perera, H., Whittle, J., Nurwidyantoro, A., Hoda, R., Shams, R. A., & Oliver, G. (2022). Human Values in Software Engineering: Contrasting Case Studies of Practice. *IEEE Transactions on Software Engineering*, 48(5), 1818–1833. <https://doi.org/10.1109/TSE.2020.3038802>
- Hussain, W., Shahin, M., Hoda, R., Whittle, J., Perera, H., Nurwidyantoro, A., Shams, R. A., & Oliver, G. (2022). How can human values be addressed in agile methods? A case study on SAFe. *IEEE Transactions on Software Engineering*, 48(12), 5158–5175.
- Jarzabkowski, P. A., Lê, J. K., & Feldman, M. S. (2012). Toward a Theory of Coordinating: Creating Coordinating Mechanisms in Practice. *Organization Science*, 23(4), 907–927.
- Jorgensen, M. (2019). Relationships between project size, agile practices, and successful software development: Results and analysis. *IEEE Software*, 36(2), 39–43.
- Kalenda, M., Hyna, P., & Rossi, B. (2018). Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process*, 30(10), e1954.
- Kettunen, P., & Laanti, M. (2017). Future software organizations – agile goals and roles. *European Journal of Futures Research*, 5(1), 16.
- Kitchenham, B. A., Dyba, T., & Jorgensen, M. (2004). Evidence-based software engineering. 273–281.
- Klein, H. K., & Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly*, 67–93.
- Kniberg, H., & Ivarsson, A. (2012). Scaling agile spotify with tribes, squads, chapters & guilds (October, 2012). URL <https://Blog.Crisp.Se/Wp-Content/Uploads/2012/11/SpotifyScaling.Pdf>.
- Kwan, I., Schroter, A., & Damian, D. (2011). Does socio-technical congruence have an effect on software build success? A study of coordination in a software project. *IEEE Transactions on Software Engineering*, 37(3), 307–324.
- Lewis, K. (2003). Measuring transactive memory systems in the field: Scale development and validation. *Journal of Applied Psychology*, 88(4), 587–604.
- Li, Y., & Maedche, A. (2012). Formulating effective coordination strategies in agile global software development teams.

- Lin, B., Zagalsky, A., Storey, M.-A., & Serebrenik, A. (2016). Why Developers Are Slacking Off: Understanding How Software Teams Use Slack. *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*, 333–336. <https://doi.org/10.1145/2818052.2869117>
- Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic inquiry*. sage.
- Malone, T. W., & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*, 26(1), 87–119.
- Marks, M. A., DeChurch, L. A., Mathieu, J. E., Panzer, F. J., & Alonso, A. (2005). Teamwork in multiteam systems. *Journal of Applied Psychology*, 90(5), 964.
- Mathieu, J. E., Marks, M. A., & Zaccaro, S. J. (2002). Multiteam systems. *Handbook of Industrial, Work and Organizational Psychology, Volume 2: Organizational Psychology.*, 289–313.
- Moe, N. B., Dingsøyr, T., & Rolland, K. (2018). To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software development. *International Journal of Information Systems and Project Management*, 6(3), 45–59.
- Molléri, J. S., Mendes, E., Petersen, K., & Felderer, M. (2023). Determining a core view of research quality in empirical software engineering. *Computer Standards & Interfaces*, 84, 103688.
- Munir, H., Wnuk, K., & Runeson, P. (2016). Open innovation in software engineering: A systematic mapping study. *Empirical Software Engineering*, 21(2), 684–723.
- Nickerson, R. C., Varshney, U., & Muntermann, J. (2013). A method for taxonomy development and its application in information systems. *European Journal of Information Systems*, 22(3), 336–359.
- Niven, P. R., & Lamorte, B. (2016). *Objectives and key results: Driving focus, alignment, and engagement with OKRs*. John Wiley & Sons.
- Nowell, L. S., Norris, J. M., White, D. E., & Moules, N. J. (2017). Thematic analysis: Striving to meet the trustworthiness criteria. *International Journal of Qualitative Methods*, 16(1), 1609406917733847.
- Nyrud, H., & Stray, V. (2017). Inter-team coordination mechanisms in large-scale agile. 16.
- Okhuysen, G. A., & Bechky, B. A. (2009). 10 coordination in organizations: An integrative perspective. *Academy of Management Annals*, 3(1), 463–502.
- Orlikowski, W. J. (1992). The Duality of Technology: Rethinking the Concept of Technology in Organizations. *Organization Science*, 3(3), 398–427. <https://doi.org/10.1287/orsc.3.3.398>
- Paasivaara, M. (2017). Adopting SAFe to scale agile in a globally distributed organization. 36–40.
- Paasivaara, M., Behm, B., Lassenius, C., & Hallikainen, M. (2018). Large-scale agile transformation at Ericsson: A case study. *Empirical Software Engineering*, 23(5), 2550–2596.
- Paasivaara, M., Heikkilä, V. T., & Lassenius, C. (2012). Experiences in scaling the product owner role in large-scale globally distributed scrum. 174–178.

- Paasivaara, M., & Lassenius, C. (2019). Empower Your Agile Organization: Community-Based Decision Making in Large-Scale Agile Development at Ericsson. *IEEE Software*, 36(2), 64–69.
- Palopak, Y., Huang, S.-J., & Ratnasari, W. (2023). Knowledge diffusion trajectories of agile software development research: A main path analysis. *Information and Software Technology*, 156, 107131. <https://doi.org/10.1016/j.infsof.2022.107131>
- Przybyłek, A., Albecka, M., Springer, O., & Kowalski, W. (2022). Game-based Sprint retrospectives: Multiple action research. *Empirical Software Engineering*, 27, 1–56.
- Ralph, P. (2018). Toward methodological guidelines for process theories and taxonomies in software engineering. *IEEE Transactions on Software Engineering*, 45(7), 712–735.
- Rico, R., Sánchez-Manzanares, M., Gil, F., & Gibson, C. (2008). Team Implicit Coordination Processes: A Team Knowledge-Based Approach. *Academy of Management Review*, 33(1), 163–184.
- Rigby, D. K., Sutherland, J., & Noble, A. (2018). Agile at scale: How to go from a few team to hundreds. *Harvard Business Review*, 96(3), 88–96.
- Rolland, K., Dingsoyr, T., Fitzgerald, B., & Stol, K.-J. (2016). Problematizing agile in the large: Alternative assumptions for large-scale agile development. 39th International Conference on Information Systems.
- Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2), 131–164.
- Russo, D. (2021). The Agile Success Model: A Mixed-methods Study of a Large-scale Agile Transformation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(4), 1–46.
- Sablis, A., Smite, D., & Moe, N. (2020). Team-external coordination in large-scale software development projects. *Journal of Software: Evolution and Process*, e2297.
- Salameh, A., & Bass, J. M. (2019). Spotify tailoring for promoting effectiveness in cross-functional autonomous squads. 20–28.
- Salameh, A., & Bass, J. M. (2022). An architecture governance approach for Agile development by tailoring the Spotify model. *AI & SOCIETY*, 37(2), 761–780.
- Scheerer, A., Hildenbrand, T., & Kude, T. (2014). Coordination in large-scale agile software development: A multiteam systems perspective. 4780–4788.
- Schmidt, K., & Simonee, C. (1996). Coordination mechanisms: Towards a conceptual foundation of CSCW systems design. *Computer Supported Cooperative Work (CSCW)*, 5(2–3), 155–200.
- Schmidtke, J. M., & Cummings, A. (2017). The effects of virtualness on teamwork behavioral components: The role of shared mental models. *Human Resource Management Review*, 27(4), 660–677. <https://doi.org/10.1016/j.hrmr.2016.12.011>
- Sebastian, I. M., & Bui, T. (2012). The influence of IS affordances on work practices in health care: A relational coordination approach (M.-H. Huang, G. Piccoli, & V. Sambamurthy, Eds.; pp. 1–11). Association for Information Systems.
- Sharp, H., Dittrich, Y., & C. R. B. de Souza. (2016). The Role of Ethnographic Studies in Empirical Software Engineering. *IEEE Transactions on Software Engineering*, 42(8), 786–804.

- Shastri, Y., Hoda, R., & Amor, R. (2021a). Spearheading agile: The role of the scrum master in agile projects. *Empirical Software Engineering*, 26(1), 3. <https://doi.org/10.1007/s10664-020-09899-4>
- Shastri, Y., Hoda, R., & Amor, R. (2021b). The role of the project manager in agile software development projects. *Journal of Systems and Software*, 173, 110871.
- Sjøberg, D. I., Dybå, T., Anda, B. C., & Hannay, J. E. (2008). Building theories in software engineering. In F. Shull, J. Singer, & D. I. K. Sjøberg (Eds.), *Guide to advanced empirical software engineering* (pp. 312–336). Springer.
- Smite, D., Christensen, E. L., Tell, P., & Russo, D. (2022). The Future Workplace: Characterizing the Spectrum of Hybrid Work Arrangements for Software Teams. *IEEE Software*.
- Smite, D., Moe, N. B., Levinta, G., & Floryan, M. (2019). Spotify Guilds: How to Succeed With Knowledge Sharing in Large-Scale Agile Organizations. *IEEE Software*, 36(2), 51–57.
- Spiegler, S. V., Heinecke, C., & Wagner, S. (2021). An empirical study on changing leadership in agile teams. *Empirical Software Engineering*, 26(3), 1–35.
- Sporsem, T., Strand, A. F., & Hanssen, G. K. (2022). Unscheduled Meetings in Hybrid Work. *IEEE Software*, 40(2), 42–49.
- Stake, R. E. (2005). Qualitative Case Studies. In N. Denzin & Y. Lincoln (Eds.), *The Sage Handbook of Qualitative Research*. Sage Publications.
- Stol, K.-J., & Fitzgerald, B. (2015). Theory-oriented software engineering. *Science of Computer Programming*, 101, 79–98.
- Stol, K.-J., & Fitzgerald, B. (2018). The ABC of software engineering research. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 27(3), 1–51.
- Storey, M.-A., Ernst, N. A., Williams, C., & Kalliamvakou, E. (2020). The who, what, how of software engineering research: A socio-technical framework. *Empirical Software Engineering*, 25(5), 4097–4129.
- Stray, V., Florea, R., & Paruch, L. (2022). Exploring human factors of the agile software tester. *Software Quality Journal*, 30(2), 455–481.
- Stray, V., & Moe, N. B. (2020). Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and Slack. *Journal of Systems and Software*, 170, 110717. <https://doi.org/10.1016/j.jss.2020.110717>
- Stray, V., Moe, N. B., & Aasheim, A. (2019). Dependency Management in Large-Scale Agile: A case study of DevOps Teams. *the 52nd Hawaii International Conference on Information Sciences*.
- Stray, V., Moe, N. B., & Noroozi, M. (2019). Slack Me If You Can! Using Enterprise Social Networking Tools in Virtual Agile Teams. 111–121.
- Stray, V., Moe, N. B., & Sjøberg, D. I. (2018). Daily Stand-Up Meetings: Start Breaking the Rules. *IEEE Software*.
- Stray, V., Moe, N. B., Strode, D., & Mæhlum, E. (2022). Coordination Value in Agile Software Development: A Multiple Case Study of Coordination Mechanisms Managing Dependencies. *Proceedings of the 15th International Conference on Cooperative and Human Aspects of Software Engineering*, 11–20. <https://doi.org/10.1145/3528579.3529182>

- Stray, V., Moe, N. B., Vedal, H., & Berntzen, M. (2022). Using Objectives and Key Results (OKRs) and Slack: A Case Study of Coordination in Large-Scale Distributed Agile. <https://doi.org/10.36227/techrxiv.16892161.v1>
- Stray, V., Sjøberg, D. I., & Dybå, T. (2016). The daily stand-up meeting: A grounded theory study. *Journal of Systems and Software*, 114, 101–124.
- Strode, D. E. (2016). A dependency taxonomy for agile software development projects. *Information Systems Frontiers*, 18(1), 23–46.
- Strode, D. E., & Huff, S. L. (2012). A taxonomy of dependencies in agile software development. 1–10.
- Strode, D. E., Huff, S. L., Hope, B., & Link, S. (2012). Coordination in co-located agile software development projects. *Journal of Systems and Software*, 85(6), 1222–1238.
- Suri, J. F. (2010). Poetic observation: What designers make of what they see. In *Design Anthropology*, Springer.
- Thompson, J. D. (1967). *Organizations in action: Social science bases of administrative theory*. McGraw-Hill.
- Tkalich, A., Ulfsnes, R., & Moe, N. B. (2022). Toward an Agile Product Management: What Do Product Managers Do in Agile Companies? 168–184.
- Turetken, O., Stojanov, I., & Trienekens, J. J. (2017). Assessing the adoption level of scaled agile development: A maturity model for Scaled Agile Framework. *Journal of Software: Evolution and Process*, 29(6), e1796.
- Uludağ, Ö., Philipp, P., Putta, A., Paasivaara, M., Lassenius, C., & Matthes, F. (2022). Revealing the state of the art of large-scale agile development research: A systematic mapping study. *Journal of Systems and Software*, 111473.
- Van de Ven, A. H., Delbecq, A. L., & Koenig Jr, R. (1976). Determinants of coordination modes within organizations. *American Sociological Review*, 322–338.
- Vedal, H., Stray, V., Berntzen, M., & Moe, N. B. (2021). Managing dependencies in large-scale agile. 52–61.
- Verne, G. B. (2021, February 1). Overview of the field and intro to research paradigms [Lecture]. <https://www.uio.no/studier/emner/matnat/ifi/IN5000/v21/lecture-1.-feb.pdf>
- Wagner III, J. A. (2023). Inter-team coordination in multiteam systems: Mechanisms, transitions, and precipitants. *Organizational Psychology Review*, 20413866231153536.
- Walsham, G. (2002). Interpretive Case Study in IS Research: Nature and Method. In M. D. Myers & D. Avison (Eds.), *Qualitative Research in Information Systems*. Sage Publications.
- Walsham, G. (2006). Doing interpretive research. *European Journal of Information Systems*, 15(3), 320–330. <https://doi.org/10.1057/palgrave.ejis.3000589>
- Walsham, G. (2012). Are we making a better world with ICTs? Reflections on a future agenda for the IS field. *Journal of Information Technology*, 27(2), 87–93.
- Weick, K. E., & Roberts, K. H. (1993). Collective mind in organizations: Heedful interrelating on flight decks. *Administrative Science Quarterly*, 357–381.
- Wenger, E., McDermott, R. A., & Snyder, W. (2002). *Cultivating communities of practice: A guide to managing knowledge*. Harvard business press.
- Wieringa, R., & Daneva, M. (2015). Six strategies for generalizing software engineering theories. *Science of Computer Programming*, 101, 136–152.

- Williams, L., & Cockburn, A. (2003). Guest Editors' Introduction: Agile Software Development: It's about Feedback and Change. *Computer*, 36(6), 39–43.
- Wohlin, C., & Aurum, A. (2015). Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering*, 20(6), 1427–1455.
- Wong, S. I., & Berntzen, M. (2019). Transformational leadership and leader–member exchange in distributed teams: The roles of electronic dependence and team task interdependence. *Computers in Human Behavior*, 92, 381–392. <https://doi.org/10.1016/j.chb.2018.11.032>
- Wong, S. I., Berntzen, M., Warner-Søderholm, G., & Giessner, S. R. (2022). The negative impact of individual perceived isolation in distributed teams and its possible remedies. *Human Resource Management Journal*.
- Yang, C., Liang, P., & Avgeriou, P. (2016). A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software*, 111, 157–184.
- Yazan, B. (2015). Three approaches to case study methods in education: Yin, Merriam, and Stake. *The Qualitative Report*, 20(2), 134–152.
- Yin, R. K. (2018). *Case study research and applications: Design and methods* (6th ed.). Sage publications.
- Zackrisson, E. J., Seibold, D. R., & Rice, R. E. (2015). Organizational coordination and communication: A critical review and integrative model. *Annals of the International Communication Association*, 39(1), 195–233.

Appendices

Appendix A: Participant consent form

Would you like to participate in the research project "Coordination in Autonomous Teams"?

This is a question to you about participating in a research project whose purpose is to contribute to new knowledge about coordination in autonomous teams in agile IT projects. In this letter, we provide you with information about the goals of the project and what participation will entail for you.

Purpose

The purpose of the project is to build new knowledge about coordination in autonomous teams in agile IT projects. By participating in this study, you contribute to research-based knowledge that is added to ICT in Norway and research internationally. We want to better understand the challenges and opportunities of coordination within and between autonomous teams working according to agile systems development methodology in various companies, and how organizations can facilitate and strengthen autonomous teams for increased innovation and productivity.

The project will investigate issues related to the overarching purpose, such as:

- What coordination mechanisms are used within and between autonomous development teams in large agile IT projects?
- What dependencies exist within and between such teams, and what challenges can be associated with these dependencies?

- What role do management processes and leadership roles play in effectiveness in agile development projects?

The project is part of a Ph.D. study conducted by Ph.D. candidate Marthe Berntzen at the University of Oslo, Department of Informatics.

Your personal information will only be used for data analysis. All results and findings that can be attributed to individual persons will be anonymized before any publication in scientific journals, use in teaching, lectures or similar.

Who is responsible for the research project?

The University of Oslo (UiO), Department of Informatics, is responsible for the project. The project is also partially affiliated with the A-Team research project at SINTEF, in that the companies affiliated with A-team are made available as potential sample companies for the Ph.D. candidate.

Why are you being asked to participate?

You are being asked to participate in the study because your employer is a participating company in the Autonomous Team (A-team) project at SINTEF. The Ph.D. study is an independent project, but the Ph.D. candidate is affiliated with the project and the companies participating through her supervisors at the University of Oslo and SINTEF.

What does it mean for you to participate?

- The project involves participatory observation in your workplace. Ph.D. candidate Marthe Berntzen will perform the observations. Field notes from the observations will be transcribed and stored electronically. If you do not want to be part of the observation notes, you can inform Ph.D. candidate Marthe Berntzen that you do not want to be referred to in the notes or to be deleted from the notes. It may also be relevant to take photographs during fieldwork, such as of the office landscape, tools used, meetings, or other things that can shed light on coordination in your workplace. You will always be asked to consent before photographs are taken.
- You may also be asked to participate in an interview, either a personal interview or a group interview with other colleagues. The themes for the interviews will be related to the overarching theme of coordination in autonomous teams and agile development. You will receive a request for a possible interview, and you can accept or decline this. Such an interview will take approximately 1 hour. The interviews will be recorded as audio recordings after your consent, or notes will be taken. The interviews will then be transcribed and stored electronically.
- You may also be asked to complete a questionnaire that will contain questions related to your experiences of topics such as teamwork, leadership, and communication in your workplace. The questionnaire will be sent to you electronically, and it will take approximately 15-30 minutes to complete.

It is voluntary to participate

Participation in the project is voluntary. If you choose to participate, you can withdraw your consent at any time without giving any reason. All information about you will then be anonymized. It will not have any negative consequences for you if you do not want to participate or later choose to withdraw.

Your privacy - how we store and use your information.

We will only use the information about you for the purposes we have described in this document. We treat the information confidentially and in accordance with privacy legislation.

- Those who will have access to your information are Ph.D. candidate Marthe Berntzen at the University of Oslo, supervisors Dr. Viktoria Stray, Associate Professor at the Department of Informatics, and Dr. Nils Brede Moe, Senior Researcher at SINTEF.
- All data collected will be treated confidentially. The data material will be secured so that unauthorized persons do not have access. Field notes and interview material will be transcribed and stored on computers locked with passwords and encryption so that it is only accessible to researchers in the project. Responses from questionnaires where personal information is registered will be protected by a separate password key and stored separately from the datasets used in the analyses.
- Questionnaires will be sent out through the Nettskjema service, which the University of Oslo has a data processing agreement with to ensure the protection of personal information.
- Participants will not be recognizable in the publication of findings and results from the data collected. The data material will be compiled and anonymized, and no analyses will be conducted on individuals or individual questions. Managers and others will only see compiled data that cannot be traced back to individuals, and all findings will first be presented to the participants in the survey.

What happens to your information when we end the research project?

The project is scheduled to end on October 31, 2024. After that, all data will be anonymized, and audio recordings will be deleted, so that the remaining data will not contain information that can contribute to the identification of participants.

Your rights

As long as you can be identified in the data, you have the right to:

- access to which personal information is registered about you,
- have personal information about you corrected,
- have personal information about you deleted,
- have a copy of your personal information (data portability), and
- to file a complaint with the privacy ombudsman or the Norwegian Data Protection Authority about the processing of your personal information.

What gives us the right to process personal information about you?

We process information about you based on your consent.

On behalf of the University of Oslo, the Department of Informatics, NSD - Norwegian Centre for Research Data AS has assessed that the processing of personal data in this project complies with the data protection regulations.

Where can I find out more?

If you have any questions about the study or wish to exercise your rights, please contact:

- University of Oslo, Department of Informatics, Ph.D. Candidate Marthe Berntzen at email marthenb@ifi.uio.no or phone 97010216.
- Our privacy ombudsman: Maren Magnus Voll (personvernombud@uio.no)
- NSD - Norwegian Centre for Research Data AS, by email (personverntjenester@nsd.no) or phone: 55 58 21 17.

Sincerely,

Project Manager (Ph.D. candidate)

Consent form

I have received and understood the information about the project "Coordination in autonomous teams" and have had the opportunity to ask questions. I consent to (tick the applicable boxes):

- participate in participant observation
- participate in personal interviews
- participate in group interviews
- participate in surveys

I consent to my information being processed until the project is completed on October 31, 2024.

(Signed by project participant, date)

Appendix B: Overview of the meeting observations

2 nd half of 2018					
No.	Name of meeting	Date	Time	Duration (min)	No. of participants
1	Product owner prioritization meeting	05.09.2018	11.30-12.30	60	14
2	Product owner weekly meeting	05.09.2018	12.30-13.30	60	8
3	Ad hoc team leader meeting	05.09.2018	13.35-13.55	30	4
4	Team Friday meeting with demo	07.09.2018	10.00-11.00	60	14
5	Product owner prioritization meeting	19.09.2018	11.30-12.30	60	14
6	Conversation with agile method specialist	28.09.2018	13.00-14.00	60	3
7	Product owner prioritization meeting	03.10.2018	11.30-12.30	60	11
8	Team Friday meeting with demo	05.10.2018	10.00-11.00	60	13
9	Testing meeting	05.10.2018	14.00-15.00	60	5
10	Team leader retrospective	08.10.2018	12.00-14.00	120	14
11	Product feature meeting	10.10.2018	10.30-11.00	30	10
12	Troubleshooting meeting	10.10.2018	13.00-13.30	30	4
13	Cake celebration	10.10.2018	14.00-14.30	30	20
14	Team Friday meeting with demo	12.10.2018	10.00-11.00	60	11
15	Product owner prioritization meeting	17.10.2018	11.30-12.30	60	14
16	PoP team retrospective	19.10.2018	09.00-11.00	120	12
17	Team leader Stand-up	29.10.2018	12.00-12.30	30	14
18	Product owner prioritization meeting	31.10.2018	11.30-12.30	60	15
19	Change workshop	02.11.2018	09.00-12.00	180	5
20	Product owner workshop	07.11.2018	08.30-12.00	210	11
21	Conversation with agile method specialist	14.11.2018		60	3
22	Product owner prioritization meeting	14.11.2018	12.00-12.15	15	12
23	Team Friday meeting with demo	16.11.2018	10.00-10.30	30	10
24	Conversation with agile method specialist	05.12.2018	12.00-13.00	60	3
25	Team Friday meeting with demo	07.12.2018		60	10
26	Change workshop	07.12.2018		210	6
27	Product owner prioritization meeting	12.12.2018	12.00-12.20	20	12
28	Product owner weekly meeting	12.12.2018	12.30-13.30	60	6
29	Team retrospective	18.12.2018	09.00-11.00	120	13
Total hours 2018				34.5	
Days on-site				18	

1 st half of 2019					
No.	Name of meeting	Date	Time	Duration (min)	No. of participants
1	Team retrospective	01.02.2019	09.00-11.00	60	11
2	Weekly program demo	01.02.2019	14.00-	20	20+
3	SubTeam 1 sprint planning	11.02.2019	09.00-09.15	15	4
4	SubTeam 2 sprint planning	11.02.2019	09.15-09.30	15	5
5	Team stand-up meeting	11.02.2019	10.45-11.00	15	10
6	Team leader stand-up meeting	11.02.2019	12.00-12.20	20	17
7	Team Friday meeting with demo	15.02.2019	10.00-11.00	60	13
8	Weekly program demo	15.02.2019	14.00-14.15	15	20+
9	Conversation with agile method specialist	18.02.2019	09.00-10.00	60	4
10	Team leader stand-up	18.02.2019	12.00-12.20	20	13
11	SubTeam 1 sprint planning	25.02.2019	10.00-10.15	15	3
12	SubTeam 2 sprint planning	25.02.2019	10.15-10.30	15	5
13	SubTeam 3 sprint planning	25.02.2019	10.30-10.45	15	4
14	Team stand-up meeting	25.02.2019	10.50-11.00	12	8
15	Team leader retrospective	25.02.2019	12.00-14.00	120	20
16	Board meeting	27.02.2019	12.30-13.30	60	9
17	Change meeting	28.02.2019	10.00-13.30	210	6
18	Party	28.02.2019	17.00-		20+
19	Team retrospective	01.03.2019	09.00-11.00	60	13
20	Weekly program demo	01.03.2019	14.00-14.25	25	20+
21	Team social night	01.03.2019	16.30-		13
22	SubTeam 4 sprint planning	04.03.2019	09.30-09.45	15	4
23	Unscheduled product feature meeting	04.03.2019	10.30-10.45	15	5
24	Team stand-up meeting	04.03.2019	10.45-11.00	15	12
25	Team leader stand-up	04.03.2019	12.00-12.20	20	14
26	Team Friday meeting with demo	22.03.2019	10.00-11.00	60	15
27	Stand-up PoP	25.03.2019	10.45-11.00	15	9
28	Team leader stand-up	25.03.2019	12.00-12.30	30	16
29	Product owner prioritization meeting	27.03.2019	11.30-11.50	20	12
30	Roadmap presentation with top management	03.04.2019	10.05-10.30	25	11
31	Client troubleshooting meeting	03.04.2019	12.00-12.45	45	7
32	Team retrospective	05.04.2019	09.00-11.00	120	15
33	Conversation with agile method specialist	05.04.2019	12.00-12.50	0,8	4
34	Inter-team lunch	09.04.2019	11.00-12.30	90	20
35	Team retrospective	10.04.2019	10.00-11.00	60	13
36	Team retrospective	23.04.2019	12.00-13.10	75	12
37	Team Friday meeting with demo	26.04.2019	10.00-10.35	35	12

38	Tech lead forum	26.04.2019	12.00-14.00	120	20-25
39	Change workshop	30.04.19	12.00-14.00	120	7
40	Unscheduled conversation	30.04.19	14.00-14.30	30	3
39	Team retrospective	03.05.2019	09.00-11.00	120	10
40	Conversation with agile method specialist	28.06.2019	11.30-12.15	45	3
41	Weekly program demo	28.06.2019	14.00-14.20	20	30+
42	Task force stand-up meeting	25.07.2019	12.30-13.00	30	6
Total hours 1st half of 2019				32.5	
Days on-site				22	
2nd half of 2019					
No.	Name of meeting	Date	Time	Duration (min)	No. of participants
43	Product owner prioritization meeting	14.08.2019	11.30-12.00	30	9
44	Client workshop pt. 1	21.08.2019	14.00-15.00	60	40-50
45	Client workshop pt. 2	21.08.2019	15.00-17.00	30	10
46	Product owner prioritization meeting	27.08.2019	14.00-14.30	60	
47	Tech lead forum	30.08.2019	12.00-14.00	120	20-25
48	Weekly program demo	30.08.2019	14.00-14.15	15	40+
49	Client workshop pt. 3	05.09.2019	11.00-12.30	90	6
50	Status meeting with client	10.09.2019	10.00-11.00	60	7
51	Team Friday meeting with demo	13.09.19	10.00-11.00	60	11
52	Tech lead forum	13.09.19	12.00-13.45	105	20-25
53	Ad hoc conversation before client retro	18.09.19	10.30-11.00	30	3
54	Client retrospective	18.09.19	13.00-15.00	120	7
55	Product owner weekly meeting	19.09.19	11.30-12.30	60	8
56	OKR workshop	26.09.19	09.00-15.30	390	25+
57	Tech lead forum	11.10.19	12.00-14.00	120	20
58	Weekly program demo	11.10.19	14.00-14.15	15	40+
59	Inter-team process meeting	11.10.19	14.15-16.00	105	6
60	Team stand-up meeting	22.10.19	10.45-10.55	10	13
61	Client onboarding meeting	22.10.19	14.30-15.50	80	3
62	Tech lead forum	25.10.19	12.00-14.00	120	20-25
63	Change workshop	31.10.19	09.30-10.50	75	6
64	Product owner weekly meeting	31.10.19	11.00-12.00	60	10
65	Product owner retrospective	14.11.19	11.30-13.00	90	8
66	Tech lead forum	25.11.19	12.00-14.00	120	25+
67	Tech lead forum	06.12.19	12.00-13.15	75	20-25
68	Product owner weekly meeting	12.12.19	14.00-15.00	60	6
69	OKR workshop	17.12.19	09.00-15.30	390	25+
Total hours 2nd half of 2019				42.5	

Days on-site					19	
January 2020						
No.	Name of meeting	Date	Time	Duration (min)	No. of participants	
1	Team stand-up meeting	14.01.20	10.45-10.55	15	11	
2	Change workshop	15.01.20	10.00-12.00	120	7	
3	Team stand-up meeting	16.01.20	10.45-11.00	15	12	
4	Product owner weekly meeting	16.01.20	11.00-12.00	60	9	
Total hours January 2020				3.5		
Days on-site					3	

Appendix C: Overview of the interviews

No.	Date	ID	Role	Gender	Tape recorded	Duration	Used in paper
1	17.10.2018	I01	Product Owner	F	Yes	22,43	1, 3, 4
2	17.10.2018	I02	Product Owner	F	Yes	29,18	1, 3, 4
3	17.10.2018	I03	Product Manager	M	Yes	28,04	1, 3, 4
4	19.10.2018	I04	Product Owner	F	Yes	40,10	1, 3, 4
5	19.10.2018	I05	Product Owner	M	Yes	32,30	1, 3, 4
6	29.10.2018	I06	CTO	M	Yes	32,49	1, 3, 4
7	29.10.2018	I07	Product Owner	M	Yes	26,04	1, 3, 4
8	29.10.2018	I08	Product Owner	M	Yes	28,30	1, 3, 4
9	29.10.2018	I09	Product Owner	M	Yes	35,15	1, 3, 4
10	29.10.2018	I10	Product Owner	M	Yes	41,24	1, 3, 4
11	29.10.2018	I11	Product Owner	F	Yes	67,31	1, 3, 4
12	29.10.2018	I12	Development Manager	M	Yes	50,46	1, 3, 4
13	23.04.2019	I13	Team leader	M	Yes	58,47	3, 4
14	25.07.2019	I14	Program Architect	M	No	65,00	3, 4
15	15.10.2019	I15	Process Specialist	M	Yes	68,11	2, 3, 4
16	16.10.2019	I16	External Architect	M	Yes	71,32	3, 4
17	17.10.2019	I17	Process Assistant	F	No	32,00	3, 4
18	25.10.2019	I18	Program Architect	M	Yes	103,11	2, 3, 4
19	30.10.2019	I19	Program Architect/Chief Architect	M	Yes	52,18	2, 3, 4
20	01.11.2019	I20	Method Specialist	M	Yes	51,09	2, 3, 4
21	13.11.2019	I06	CTO	M	Yes	47,31	2, 3, 4
22	13.11.2019	I14	Program Architect	M	Yes	86,54	2, 3, 4
23	13.11.2019	I03	Product Manager	M	Yes	59,08	2, 3, 4
24	15.11.2019	I12	Development Manager	M	Yes	57,38	2, 3, 4
25	14.01.2020	I21	Tech lead	M	Yes	54,47	2, 3, 4
26	14.01.2020	I22	Tech lead	M	Yes	56,24	2, 3, 4
27	14.01.2020	I23	Tech lead	M	No	65,00	3, 4
28	16.01.2020	I24	Tech lead	F	Yes	38,41	2, 3, 4
29	16.01.2020	I25	Team leader	M	Yes	47,36	2, 3, 4
30	20.01.2020	I18	Program Architect	M	No	65,00	3,4
31	20.01.2020	I15	Process Specialist	M	No	55,00	3,4
32	28.09.2018	I20	Method Specialist	M	No	45,00	4
33	14.11.2018	I20	Method Specialist	M	No	60,00	4
34	05.12.2018	I20	Method Specialist	M	No	60,00	4
35	18.02.2019	I20	Method Specialist	M	No	55,00	4
36	05.04.2019	I20	Method Specialist	M	No	50,00	4
37	27.06.2019	I20	Method Specialist	M	No	45,00	4
Mean length (minutes)						52,50	

Paper 1: The Product Owner in Large-Scale Agile: An Empirical Study through the lens of Relational Coordination Theory

Marthe Berntzen, Nils Brede Moe, and Viktoria Stray

In Proceedings of the International Conference on Agile Software Development (pp. 121-136). Springer, Cham, 2019

Abstract

In agile software development, a core responsibility of the Product Owner (PO) is to communicate business needs to the development team. In large-scale agile software development projects, many teams work towards an overall outcome, but also need to manage interdependencies and coordinate efficiently. In such settings, the POs need to coordinate for shared knowledge about project status and goal attainment within, but also across, development teams. Previous research has shown that the PO assumes a wide set of roles, however, our knowledge about how POs coordinate in large-scale agile is limited. In this case study, we explore this in a large-scale development program through the theoretical lens of Relational Coordination Theory. Our findings suggest that 1) Coordination varies depending on the context of each PO, 2) the focus on achieving high-quality communication changes coordination over time, and 3) Unscheduled coordination is an enabler for high-quality communication.

1 Introduction

Coordination is key to large-scale agile software development projects [4, 6]. In large-scale agile projects, the number of interdependencies requires the collective input of multiple teams and individuals, often with nonoverlapping knowledge sets. Because of frequent changes, size and complexity, large-scale agile projects have a high level of uncertainty. In such high uncertainty contexts, it is more important to control output (e.g., by setting goals and targets) rather than to control behavior (e.g., through rules and programs). This can be achieved by relying on continuous feedback and mutual adjustment [16]. Further, the high levels of uncertainty and dependencies in large agile projects require subcentral unscheduled coordination and the need for coordination mechanisms to continually emerge [18]. Additionally, delivering value frequently requires work and knowledge coordination on

different levels, e.g., the program, project, and team levels. Teams need to manage dependencies with other teams, experts, managers and stakeholders [22]. To achieve effective coordination, participants must be connected by relationships of shared goals and mutual respect [11, 12].

Inter-team coordination is one mechanism for manage dependencies in large-scale agile. Dingsøyr, et al. [6] describe 14 inter-team coordination mechanisms in a large-scale software project while Stray et al [24] identified 20 mechanisms (eleven synchronization activities and nine synchronization artifacts). Paasivaara et al. [20] found that the Product Owner (PO) and the PO team were critical in assisting with inter-team coordination. To understand coordination in large-scale agile, the PO role and coordination mechanisms related to this role is crucial to understand. To the best of our knowledge, the existing literature does not address how POs coordinate work within and across teams in large-scale agile.

Motivated by the importance of coordination in large-scale agile and need to understand the coordination in PO teams, our research question is: *How do Product Owners coordinate work in large-scale agile?* The remainder of the paper is organized as follows. Section 2 outlines related work. In Section 3, we describe our research methodology. In Section 4, we present our findings from the cases, further discussed in Section 5 which also concludes the paper with a summary of major findings.

2 Background and related work

2.1 The Product Owner role in large-Scale Agile

Agile approaches focus on self-management, emergent processes, and informal coordinating mechanisms. The software team achieves coordination by the simple process of informal communication [8]. When scaling up agile, several challenges arise, such as managing a larger number of stakeholders, keeping to the agile principles and coordinating the different teams while keeping an informal approach to communication [4-6, 8]. In large software projects, informal communication can take part within teams, between group of managers or groups of representatives acting on behalf of their teams.

The PO role is defined in Scrum as a person who gather and prioritize requirements and interacts with the customer [21]. A PO need to understand what should be developed and translate and communicate these business needs to the development team [1]. The PO define and prioritize the features of the product, decide on release date and content and are responsible for the profitability of the product [25]. The development team is responsible for designing, testing and deploying systems, while the PO knows what system to be built.

In large-scale agile, POs form teams to be able to gather and prioritize inter-team requirements in the face of conflicting and competing business needs [1]. The POs in these teams can either share responsibility or be responsible for a subset of product features [20]. Bass [1] identified nine different functions that product owners have in large-scale projects which included architectural coordination, assessing risk, and ensuring project compliance with corporate guidelines and policies. As such, the PO role is a complex role with a broad set of

responsibilities, which in large-scale settings may need to coordinate complex, interdependent tasks and team goals contributing to the overall goals of the software project.

2.2 Relational Coordination Theory

Relational Coordination Theory (RCT) is an established and empirically validated theory that originated from research conducted in the airline industry in the 1990s [11]. RCT holds that relationships are central to coordination towards common outcomes. An assumption of RCT is that relational coordination is stronger in more horizontally designed organizational structures [13], which is important to large-scale agile [7, 19].

Relational coordination is defined as “a mutually reinforcing process of interaction between communication and relationships carried out for the purpose of task integration” [12]. Gittell [11] propose that relationships provide the necessary bandwidth for coordinating work highly interdependent, uncertain and time-constrained settings, and that effective coordination in these settings is carried out through relationships of *shared knowledge*, *shared goals* and *mutual respect*, described in the below sections. These three concepts are mutually facilitated by frequent, timely, accurate and problem-solving communication [10, 11]. Because large-scale agile are characterized by high levels of interdependence, uncertainty and time pressure, and because autonomy is a central tenet in large-scale agile [4], we believe RCT is an interesting theoretical lens for studying coordination in large-scale agile.

Shared knowledge informs participants of how their tasks, as well as the tasks of others, contribute to the overall work process [11]. However, individuals and groups working on different functional tasks often reside in different “thought worlds”, which can hamper effective coordination because of the lack of insight about others’ work [9]. Drawing on sensemaking theory [28] and transactive memory theory [15], RCT suggests that a shared understanding of the work process and a common understanding of each other’s areas of expertise across roles facilitates the coordination of knowledge [11]. When participants know how their tasks fit other tasks in the work process, they will better understand who will be impacted by changes, in other words, they will understand who needs to know what, why and when [10].

In large-scale system development no one can know everything. Therefore, teams’ and peoples’ knowledge networks are essential. Šmite and colleagues [22] found the size of teams’ knowledge networks in a large-scale agile company to be dependent on the number of years the individual team members had been in the company in addition to which forums the individual participated in.

Shared goals. A goal may be seen as shared to the extent that employees across functional areas are aware of the same goals and have a similar understanding of why they are important [11, 26]. Thus, they play an essential role for effective coordination through enabling people to accomplish a set of complex interdependent tasks [26, 27], a common characteristic in large-scale agile software development projects where autonomous teams work on different parts of an overall product.

In large-scale agile, the collective goal of the project or program can be broken down into a goal hierarchy. The goal hierarchy is important for teams in large-scale agile to share a distal

goal while the individual teams pursue their more proximal goals. Nyrud and Stray [19] found that the Demo meeting and Backlog grooming was essential in this context because it provided an arena for creating common expectations and understanding the finished product – shared goals within and outside of the team. Moe et al., [17] found that when managers set goals in a large-scale project without involving the team, it resulted in team members being uncertain about the goal of the project.

Mutual respect. Finally, for effective coordination to occur, employees should be connected by relationships of mutual respect between the coordinating parties. According to RCT, mutual respect reinforces the inclination to act in accordance with the overall work process by establishing a middle ground [11].

A study of large-scale Scrum found that responding respectfully to each other fostered psychology safety, which is important for agile teams [23]. In a study of a large-scale project Moe et al., [17] found that external stakeholders approach team members directly, despite members expressing that it disrupts the work. Bypassing the established process reduced team progress.

Table 1. Elements of relational coordination based on the synthesis by Gittell [11].

Relational Coordination	Definition	Specific examples
Shared knowledge	Informs participants of how their own and others' tasks contribute to the overall work process. A shared understanding of the work process and others' areas of expertise facilitate knowledge coordination.	Knowledge about overall delivery milestones; Knowledge about which team is working on what, when.
Shared goals	Direct the attention and effort of individuals and groups. Transcend functional goals of different work units and enable unified effort towards a collective outcome.	Keeping in mind overall program goals while working on team goals.
Mutual respect	Valuing others' contributions and consider the impact of their own actions to the work of others.	Consider the impact of one teams' work on another; Acknowledge differences in priorities; Trust others decisions and work.
High-quality communication	Communication that is frequent, accurate, timely and problem-solving in nature.	Keep meetings relevant, send and receive information at the right time with the right content; constructive feedback

3 Method

We chose a case study approach [29] because there is little knowledge about how POs coordinate work in large-scale agile. Case studies provide depth and detailed knowledge about the case. We selected a case where almost the whole development program was co-located in

order to reduce the effects due to the distribution of teams. In the following, we refer to the case as the PubTrans program. The program started in 2016 and aim to develop a new platform supporting public transportation. The first author conducted fieldwork at the program and was given access to rich sources of data, including meetings, Slack³ channels and documentation tools. In addition, the two other authors participated in site visits, workshops and in two of the interviews.

3.1 Case description

The PubTrans program has thirteen development teams ranging between five and fourteen team members working towards the same products, and can thus be classified as very large-scale agile [6]. In order to coordinate work within and across teams the program make use of various electronic tools, such as Slack, Jira and Confluence, material artefacts such as task boards, and various scheduled and unscheduled meetings. The development teams are autonomous to the extent that they may choose freely how they go about solving their tasks, and rely on agile methods of choice. As such, there is no one unified agile approach across the teams. The POs are situated within each team. Seven of the POs have one team whereas two have three teams each. The PO's have a varied background, some have a technical (e.g. engineering) background and has been working in the product domain for several years, whereas others came from industries such as marketing and business development.

3.2 Data collection

We conducted twelve interviews in October 2018. The interviews were semi-structured and we allowed the conversations to develop naturally as the participants unfolded their stories. The duration of the interviews was between 30 and 60 minutes (average of 40 minutes), they were tape-recorded based on the participants' consent, and were later transcribed. We spent a total of eighteen days on-site with participant observation. Further, we participated in several PubTrans activities, described in Table 2.

Table 2. Data sources

Data	Description
Interviews	9 PO's, 1 Product Manager, 1 Development Manager and 1 CTO.
Observations	7 PO task board meetings (update task progress and discuss activities across teams), 2 PO weekly meetings, 5 weekly status meetings in one of the 13 teams, and 2 retrospectives in one team and one team leader retrospective. We facilitated one retrospective meeting during a PO workshop
Documents	Analyzed Slack logs from PO channel and one team channel

³ Slack is an electronic communication tool, trademark of Slack Technologies, www.slack.com.

3.3 Data analysis

When analyzing the data material, we relied on data triangulation, including observation, interviews and documentation as data sources (see Table 2). Our rationale for the choice of these data sources for the study of PO coordination is that by interviewing the participants, we gain access to their own understanding of their work routines. Analyzing the observations and documentation such as Slack logs shed light on the accounts given by the interviewees and provide context to their statements. As such, data triangulation is likely to contribute to strengthening our findings and conclusions by increased accuracy and compellability [29].

Through our engagement with the data, RCT emerged as an appropriate lens for examining product owner coordination in a large-scale agile setting. This is because RCT is a suitable theory for organizational contexts characterized by high levels of interdependence, outcome uncertainty and time criticality [10, 11] typical in large-scale development. We coded the data with NVivo according to the coordination mechanisms used by the POs (Table 3), and how these mechanisms related to the RCT concepts defined in Table 1.

4 Results

Table 3 shows the main coordination mechanisms involving the POs. In the following, we describe a selection of these coordination mechanisms in relation to the relational coordination concepts of shared goals, shared knowledge, mutual respect and high-quality communication.

4.1 Coordination between POs

The weekly PO coordination meeting enabled discussions on shared experiences and matters that came up during the past week. For instance, POs discussed challenges with team processes or updated each other on external client issues. Having a weekly meeting contributed to communication that was problem-solving, accurate and frequent. However, its content seemed to vary. One PO told us that *“There is no fixed, no defined agenda. We are supposed to talk about what is on our mind, and that is a very open question! [laughs]. It can be anything. So I think there have been some meetings that we have not gained so much from”*.

Table 3. Product Owner coordination mechanisms

Coordination mechanism	Between PO	PO and team
Product Owner weekly meeting	x	
Product Owner task board meeting	x	
Product Owner workshop activities	x	
Unscheduled conversations	x	x
Slack	x	x
Jira	x	x
Confluence/wiki	x	x
Team status meetings		x
Team retrospectives		x
Internal team practices		x

The POs expressed different opinions toward this weekly meeting. Some POs thought it was very useful, in particular for building shared knowledge and goals. Some thought that once a week was too frequent, as they wanted to spend more time with their team, whereas others felt there should be more PO meetings like this, as “*we don’t have any places to meet to exchange experiences across teams, other than these Product Owner meetings*”. As such the meeting appeared to be an important coordination mechanism in relation to shared knowledge and goals.

The bi-weekly task board meeting gathers the PO’s and relevant stakeholders such as the Product Manager, CTO and program management. All meet in front of a large visual task board (Fig. 1) to update each other on their progress - current in-progress tasks and long-term delivery milestones. As such, this artefact provides all POs with shared knowledge of current goals and status of the teams’ different tasks. The task board meeting was initially termed “prioritization meeting”, but according to the participants, this meeting did not meet its purpose. Rather than focusing on task prioritization, it was more of a reporting and updating meeting where all POs simply reported on their teams’ progress, and many talked for several minutes about their teams’ internal tasks. Until recently, the meeting lasted one hour, and we observed how the POs struggled to pay attention to what others were saying as time went by. One PO said; “*If we compare hours spent [at this meeting] versus insights gained, it doesn’t add up*”. Across several meetings, we observed that several sat down on the floor after a while, some started looking at their phones, responding to messages and emails, rather than listening. During the interviews, some said that they felt bad for being disrespectful when they did not pay attention. Most of them, however, perceived the intention behind the meeting – updating each other on progress across teams – as useful and they therefore wanted to keep the meeting, but in a different format.



Fig. 1. The PO task board meeting.

Unscheduled coordination between POs was common and was done just by walking over to each other in the open landscape. One PO explained “I seek out people at their desks... It is something about it, one thing is to communicate in writing [e.g., sending an email], but in my experience, you accomplish more by just talking to people”. Sometimes a PO would also call in to a spontaneous meeting inviting only those that needed to be part of the particular coordination activity.

Moreover, the POs have a dedicated Slack channel, created in March 2018, for knowledge sharing and quick updates regarding goal attainment from the different POs’ teams. During the interviews, it became clear that this channel was used to varying degrees by the different POs. Primarily, it was used for frequent and timely information updates such as notifying each other of absence, or uploading documents such as plans and presentations, rather than for knowledge sharing and ensuring the attainment of shared goal across teams. For instance, one particular day in September 2018, two POs discussed whether to hold the weekly PO meeting:

PO 1: *“Does this mean you will not be here today either @PO 3? [PO 4] said he too would be absent today, and so is [the Product Manager], and when there is so few of us, is there a point in going through the things we agreed all of us should be part of? Should we skip the meeting?”*

PO 2: *“I’d like to meet those of you who are present, but we can postpone the planned common PO discussion theme”*

PO 1: *“We’ll meet as planned then.”*

The POs also use the Slack channel more informally. During a PO workshop, one posted “This is the smallest hotel room I ever saw! You’ll find me in the bar”. This social and informal communication may indicate mutual respect and a sense of community among the POs, a mutual respect that is perhaps reinforced by the coordination activities they perform throughout the year.

The PO quarterly workshop gathers the POs normally overnight at an off-site location. Prior to these quarterly workshops, all POs attend a set of preparation meetings. This is done to gain a sense of shared goals and knowledge before the workshop in order to work more efficiently together. As such, the quarterly workshop both contribute to shared knowledge and shared goals between POs at an overall program level, but it may also reinforce mutual respect between the POs as they get to know each other better. As stated by one of the POs: *“It is... both professionally useful, but it’s also about getting together. It is rather social, actually”*. The topics of the workshops depend on upcoming issues in the PubTrans program, for instance discussing the potential implications of overall program strategies in relation to specific team and cross-team deliveries the upcoming quarter. Another theme could be improving their own work processes, such as inter team coordination.

In late fall 2018, two of the authors joined the POs for one such workshop with the Product Manager and eight of the nine POs. At this workshop, we facilitated a retrospective with the POs focusing on coordination efficiency. One outcome of this retrospective was four actions points they believed would improve the coordination. First, in relation to the quarterly PO workshop, some POs expressed that they would prefer if the workshops were one full workday, with no over-night stay as some felt it took up too much time. This led to some discussion, but

eventually, while other POs appreciated the change of scenery, they agreed to try the next workshop as a one-day workshop. This demonstrates that although the POs do not always agree or have the same preferences, they are willing to adjust to each other, which may indicate mutual respect.

A second action point was to move all written communication to Slack rather than use it as a supplement to e-mail. After the workshop, we observed a change in the communication in the dedicated PO Slack channel. Communication became more frequent as the PO used it more, and contributed more towards shared goals and knowledge among the POs. For instance, the POs started to share “best practice” tips and work routines on Slack, as well as agenda points for the weekly PO meeting. A third action point was to increase the focus on a clearer agenda for the weekly PO meeting. As such, the communication at this meeting may have become more accurate, which in turn contribute to reinforcing shared knowledge and goals. Finally, the fourth action point was to reduce the length of the task board meeting from one hour to 20 minutes and to focus only on updates relevant for at least two thirds of the attendants. In the following three meetings we observed that the new format lead to communication that was more accurate and timely.

4.2 Coordination between POs and their teams

We found differences between how the POs coordinate with their teams. Some POs have well-established practices and close, regular interaction with the team, while others have a more loosely defined approach with a high level of delegation.

Coordination with the team leader was a key process for most POs. Several POs spoke respectfully about their team leaders, seeing them as having both good people skills and technical skills. The POs described the team leader as an essential link for coordination with the team that often joined the PO in the decision-making. One PO explained: *“We go through all priorities together. [...] we are rarely in disagreement. And if there is... it could, for instance, be that I have knowledge from the business side that calls for different priorities, then I make the decision, but normally we agree.”*

During a team retrospective, several team members expressed the importance of the PO and the team leader in shielding the team from external pressure, and in making sure they knew which tasks to work on. This may indicate both the importance of these roles in relation to shared knowledge, and what many POs found important; respect for the developers’ time and their role towards the overall goal attainment.

Stand-up meetings with the team varied in frequency. Some teams had stand-up meetings every day, others once or twice a week, and some on a more ad hoc basis, for instance through sharing task-related information in team Slack channels. The stand-up meeting was an important meeting for sharing knowledge and solving issues. A PO explained the challenge of just listening and then being an active participant in the meeting: *“I want to be part of the stand-ups, as I want to pay attention to what they are doing [...] but then they expect me to say something, and I feel that I have to, otherwise it is all ‘top-down’”*. He further explained that he wanted to listen and learn from the team, but at the same time, he was not sure what he could

bring into the meeting, as he saw his work tasks as very different from the team's and did not find it relevant to talk about those tasks.

Retrospectives with the team varied in frequency and process. When the team members got together to discuss their work over the last period of time, the meeting contributed to strengthening shared knowledge about the teamwork processes and shared team goals in that the teams analyze, discuss and adjust their own practices. Mutual respect among the participants might also be strengthened as they share their thoughts and perspectives. Many POs left it up to the team leader to facilitate team retrospectives, while some take a more active role.

Further, the retrospectives gave important information as to how the POs may adjust their coordination practices towards the team. A PO explained: *“I thought me and the team leader were good at bringing information back to the developers. As it turned out during our last retrospective...we were not! And we are going to do something about that.”* This illustrates the importance of conducting retrospectives so that the team may mutually adjust to better accommodate each other. The willingness to adjust based on feedback may also indicate respect towards the team members through acknowledging the impact a lack of information may have on their work.

Unscheduled coordination with the team appeared important for fast decision-making. Much of the coordination with the team occurred during spontaneous conversations and meetings, and many decisions at the team level were made during such unscheduled conversations. According to one PO *“If there are decisions to be made in relation to choice of technology or similar, normally it would be me, the team lead and some developer... we just decide then and there [...]”*. This illustrates how shared knowledge about decisions are reached through accurate, timely, problem-solving communication.

Slack was also extensively used among the teams in the PubTrans program; almost all teams appeared to have closed private channels where the whole team, including the PO, discussed internal matters. In addition, there was a range of public channels for different topics. While Slack was seen as an invaluable source of knowledge and information, for some it became too much. One PO of three teams explained *“I spent some time adjusting from e-mail to Slack. [...] There are so many channels! It is so much to pay attention to and read, it can actually be a bit too much”*. The same PO further explained that Slack was not used for making larger decisions, but that overall, Slack was a great place to keep the discussion going on technical issues and every-day work-related matters.

5 Discussion and conclusion

The Product Owner is an important role in agile development, often performing a complex set of roles [1]. Our findings underscore the importance of relationships for efficient coordination among POs and between the PO and the team. We have attempted to shed light on PO coordination through the concepts of RCT. We now turn to discuss our research question *“How do Product Owners coordinate work in large-scale agile?”*.

Our analysis of PO coordination in a large-scale agile development company show that 1) coordination varies depending on the context of each PO (type of team, experience,

preferences), 2) a focus on high-quality communication changes coordination over time, and 3) unscheduled coordination enable high-quality communication.

5.1 Coordination practices varies between the POs

During our observations and in the interviews, we noticed several differences in PO coordination both among each other and towards their teams. This may be due to differences in coordination preferences among the POs, but it may also be due to the autonomy the teams have in choosing their approach to agile methods, leading to a variety in coordination mechanisms between the POs and their teams. Further, the number of teams the POs are responsible for may influence their coordination practices, as the more teams they have, the more coordination is needed.

While it appears that all coordination practices between the POs contribute to relational coordination through emphasizing shared knowledge, goals and mutual respect, not everyone got the same benefits out of the coordination mechanisms. This may be related to communication frequency, where the POs that have more frequent, timely and accurate communication with their fellow POs and their teams may have a greater understanding of the knowledge and goals in the PubTrans program. Our finding is consistent with RCT in that high-quality communication reinforces shared goals and knowledge [11-13], and that communication networks matter [10]. In relation to this, Šmite et al. [22] found that the frequency of communication and the number of actors a person coordinated with depended on how long the person had been in the company. The longer experience, the more frequent communication, which indicate that coordination becomes more accurate because of knowledge about who knows what.

5.2 Changes in coordination over time

Several of the coordination mechanisms involving the POs emerged and changed during the period of the study. Our findings are consistent with those of Jarzabkowski et al. [14], who argue that coordinating mechanisms do not appear as ready-to-use techniques but are formed as actors go about the process of coordinating. Further, coordinating mechanisms are not stable entities, but emerge through their use in ongoing interactions [14].

Throughout our period of data collection, we observed how PubTrans focused on improving coordination. During the retrospective, several action points were set, and we observed how coordination mechanisms such as the task board meeting, was improved by more timely and accurate communication in that the meeting became shorter and more focused. We also observed a change in the PO Slack channel towards more frequent and problem-solving communication, for instance by using the channel for sharing agenda points for meetings and sharing best practices from teams.

5.3 Unscheduled and frequent coordination enables high-quality communication

In the PubTrans program, practices such as the task board meeting between POs and stand-up meetings with the teams appeared essential for coordination. However, the differences in

routines in each team may make it more challenging to coordinate across teams, and to ensure a shared understanding of goals among the POs and across and the different teams.

As a supplement to the scheduled meetings, we find that unscheduled meetings appeared to be an important driver of high-quality communication in the PubTrans program used to coordinate on a daily basis. Such unscheduled conversations and meetings contribute to strengthening the shared knowledge and goals, and can be seen as timely and problem-solving communication, in particular when only a subset of the POs need to coordinate. It also indicates respect for the others' time by not including more people than necessary. In line with our findings, Dingsøy, et al. [6] found the importance of communication in large-scale agile to be both informal and formal, happening both in groups and by two people meeting. Further, they found that an open work area in large-scale agile supported fast communication in informal meetings. Our results support these notions in that unscheduled meetings and seeking out people at their desks appear important for efficient day-to-day coordination. We also find that the use of Slack enables timely and frequent and unscheduled coordination between subsets of people, such as between the POs or within teams. As such, our results indicate that standardizing the communication channels by having one digital platform that is prioritized by everyone contributes to shared knowledge across POs and teams. Finally, according to RCT, relationships between roles are central for coordination [10-12]. Our results indicate that frequent coordination between the PO and the team leader is key for high-quality communication, knowledge sharing and updates about goal attainment with the teams.

5.4 Implications for theory and future research

As can be derived from our results and this discussion, the elements of RCT are evident in the coordination mechanisms used by the POs in the PubTrans program. The theory, therefore, appears suitable for studying coordination in a large-scale agile setting. In RCT, organizational change is seen as intertwined with the relationships between roles and high-quality communication reinforcing shared goals, knowledge and respect. Research that explores organizational change to further develop the theory has been encouraged [10]. While this study contributes to the understanding of changes in coordination over time, this study is the first to utilize RCT in large-scale agile for understanding PO coordination. In her work from the airline and health industries, which also represent large-scale settings, Gittell [11, 12] observed differences between companies in how successful their coordination efforts were. Her research revealed that the companies that performed best, had better coordination between roles, and that this was related to the extent of shared knowledge and goals as well as mutual respect. Therefore, there is a need for more studies from other large-scale agile programs to better understand our findings.

5.5 Implication for practice

We believe that our study has the following main implications for product owner coordination in large-scale agile. First, we recommend focusing more on unscheduled meetings rather than scheduled, time-consuming meetings, as also suggested by other research on large-scale agile [18, 24]. Unscheduled meetings enable spontaneous coordination that contribute to shared goals, shared knowledge and mutual respect in large-scale agile. Such meetings are facilitated

by open workspace and co-located teams. Second, we recommend agreeing on a common communication infrastructure, such as Slack, for swift communication and information sharing, but also for the POs to have its own space where they can discuss outside of the scheduled meeting arenas. Third, frequent meetings and workshops where POs may discuss goals and share knowledge are necessary. However, such meetings should have a clear, predefined agenda in order to ensure efficient use of time and resources. Forth, scheduled workshops throughout the year contribute both to professional growth and forming social bonds supporting relational coordination. Finally, we advise regular retrospectives focusing on improving coordination, strengthening shared knowledge and goals, and reinforcing mutual respect and trust within the PO group.

5.6 Limitations and concluding remarks

A limitation of our research is the reliance on a single case. As such, the general criticisms of single-case studies [3, 29], apply to our study. However, our rationale for choosing the PubTrans program as our case was that it represents a setting where large-scale agile has been applied since the outset of the program in 2016. Further, the program is largely co-located and the POs are considered part of the teams, the case provided a unique setting for exploring how POs coordinate in large-scale agile settings. A further limitation relates to the reliance on semi-structured interviews as a major source of data collection and analysis [3]. However, data triangulation made it possible to study the phenomena of interest from different viewpoints, also during the changes we observed, which should serve to strengthen our results [29]. We facilitated a PO retrospective where concrete action points were formed, indicating that we did affect how work processes are conducted at the PubTrans program, at least for the time being. However, the PubTrans program had a high awareness of challenges with inter-team coordination since before we started our research. Therefore, we do not believe our presence has biased the results.

On a concluding note, in this paper, we have applied a relational coordination lens to the question of how Product Owners coordinate work in large-scale agile system development. Our findings suggest that the PO contributes to shared knowledge and goals both within and across teams in large-scale agile, and that efficient coordination also includes relationships of mutual respect and high-quality communication. This is in line with previous findings from research on RCT, however, this study is the first to investigate relational coordination in a large-scale system development setting. As such, this study makes way for future research that can contribute both to the further development of RCT as well as improving our understanding of coordination in large-scale agile development.

Acknowledgements

This research was supported by the Research Council of Norway through the research project Autonomous teams (A-team) project, under Grant Number 267704.

References

1. Bass, J. M., "How product owner teams scale agile methods to large distributed enterprises," *Empirical Software Engineering*, vol. 20, no. 6, pp. 1525-1557, 2015.
2. Begel, A., Nagappan, N., Poile, C., and Layman, L., "Coordination in large-scale software teams," in *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, 2009, pp. 1-7: IEEE Computer Society.
3. Diefenbach, T., "Are case studies more than sophisticated storytelling?: Methodological problems of qualitative empirical research mainly based on semi-structured interviews," *Quality & Quantity*, vol. 43, no. 6, p. 875, 2009.
4. Dikert, K., Paasivaara, M., and Lassenius, C., "Challenges and success factors for large-scale agile transformations: A systematic literature review," *Journal of Systems and Software*, vol. 119, pp. 87-108, 2016.
5. Dingsøy, T. and Moe, N. B., "Towards principles of large-scale agile development," in *Proceedings of the International Conference on Agile Software Development*, 2014, pp. 1-8: Springer.
6. Dingsøy, T., Moe, N. B., Fægri, T. E., and Seim, E. A., "Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation," *Empirical Software Engineering*, vol. 23, no. 1, pp. 490-520, 2018.
7. Dingsøy, T., Moe, N. B., and Seim, E. A., "Coordinating Knowledge Work in Multi-Team Programs: Findings from a Large-Scale Agile Development Program," *Project Management Journal*, vol. 49, pp. 64-77, 2018.
8. Dingsøy, T., Nerur, S., Balijepally, V., and Moe, N. B., "A decade of agile methodologies: Towards explaining agile software development," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1213-1221, 2012.
9. Dougherty, D., "Interpretive barriers to successful product innovation in large firms," *Organization science*, vol. 3, no. 2, pp. 179-202, 1992.
10. Gittell, J. H., "New Directions for Relational Coordination Theory," in *The Oxford Handbook of Positive Organizational Scholarship*, G. M. Spreitzer and K. S. Cameron, Eds.: Oxford University Press, 2012.
11. Gittell, J. H., "Relational coordination: Coordinating work through relationships of shared goals, shared knowledge and mutual respect," *Relational perspectives in organizational studies: A research companion*, pp. 74-94, 2006.
12. Gittell, J. H., "Relationships between service providers and their impact on customers," *Journal of Service Research*, vol. 4, no. 4, pp. 299-311, 2002.
13. Gittell, J. H. and Douglass, A., "Relational bureaucracy: Structuring reciprocal relationships into roles," *Academy of Management Review*, vol. 37, no. 4, pp. 709-733, 2012.
14. Jarzabkowski, P. A., Lê, J. K., and Feldman, M. S., "Toward a Theory of Coordinating: Creating Coordinating Mechanisms in Practice," *Organization Science*, vol. 23, no. 4, pp. 907-927, 2012.
15. Liang, D. W., Moreland, R., and Argote, L., "Group versus individual training and group performance: The mediating role of transactive memory," *Personality and Social Psychology Bulletin*, vol. 21, no. 4, pp. 384-393, 1995.
16. Mintzberg, H., *Mintzberg on management: Inside our strange world of organizations*. Simon and Schuster, 1989.
17. Moe, N. B., Dahl, B., Stray, V., Karlsen, L. S., and Schjødt-Osmo, S., "Team Autonomy in Large-Scale Agile," in *Proceedings of the 52nd Hawaii International Conference on Information Sciences*, 2019.
18. Moe, N. B., Dingsøy, T., and Rolland, K., "To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software development,"

-
- International Journal of Information Systems and Project Management*, vol. 6. no. 3, pp. 45-59, 2018.
19. Nyrud, H. and Stray, V., "Inter-team coordination mechanisms in large-scale agile," in *Proceedings of the XP2017 Scientific Workshops*, 2017, p. 16: ACM.
 20. Paasivaara, M., Lassenius, C., and Heikkila, V. T., "Inter-team coordination in large-scale globally distributed scrum: do scrum-of-scrums really work?," in *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, , 2012.
 21. Schwaber, K. and Beedle, M., *Agile software development with Scrum*. Prentice Hall Upper Saddle River, 2002.
 22. Šmite, D., Moe, N. B., Šāblis, A., and Wohlin, C., "Software teams and their knowledge networks in large-scale software development," *Information and Software Technology*, vol. 86, pp. 71-86, 2017.
 23. Stray, V., Fægri, T. E., and Moe, N. B., "Exploring norms in agile software teams," in *Proceedings of the International Conference on Product-Focused Software Process Improvement*, 2016, pp. 458-467: Springer.
 24. Stray, V., Moe, N. B., and Aasheim, A., "Dependency Management in Large-Scale Agile: A case study of DevOps Teams," in *Proceedings of the 52nd Hawaii International Conference on Information Sciences*, 2019.
 25. Sutherland, J. and Schwaber, K., "The scrum papers: Nuts, bolts, and origins of an agile process," 2007.
 26. Saavedra, R., Earley, P. C., and Van Dyne, L., "Complex interdependence in task-performing groups," *Journal of applied psychology*, vol. 78, no. 1, p. 61, 1993.
 27. Wageman, R., "Interdependence and group effectiveness," *Administrative science quarterly*, pp. 145-180, 1995.
 28. Weick, K. E., "The collapse of sensemaking in organizations: The Mann Gulch disaster," *Administrative science quarterly*, pp. 628-652, 1993.
 29. Yin, R. K., *Case study research: design and methods*. Thousand Oaks: Sage, 2002.

Paper 2: Coordination Strategies: Managing Inter-team Coordination Challenges in Large-Scale Agile

Marthe Berntzen, Viktoria Stray, and Nils Brede Moe

In Proceedings of the International Conference on Agile Software Development (pp. 140-156). Springer, Cham, 2021

Abstract

Inter-team coordination in large-scale software development can be challenging when relying on agile development methods that emphasize iterative and frequent delivery in autonomous teams. Previous research has introduced the concept of coordination strategies, which refer to a set of coordination mechanisms to manage dependencies. We report on a case study in a large-scale agile development program with 16 development teams. Through interviews, meeting observations, and supplemental document analyses, we explore the challenges to inter-team coordination and how dependencies are managed. We found four coordination strategies: 1) aligning autonomous teams, 2) maintaining overview in the large-scale setting, 3) managing prioritizations, and 4) managing architecture and technical dependencies. This study extends previous research on coordination strategies within teams to the inter-team level. We propose that large-scale organizations can use coordination strategies to understand how they coordinate across teams and manage their unique coordination situation.

Keywords: Coordination Strategies, Coordination Mechanisms, Dependency Management, Large-Scale Agile, Inter-team Coordination, Software Development.

1 Introduction

Digital transformation drives new sectors, such as the finance and transportation sectors, to make use of agile development methods, often in large-scale settings. Despite the popularity of agile, there are new and complex challenges associated with agile methods in large-scale settings due to the unavoidable coordination required when many development teams work together [1–3]. When many teams work simultaneously with large code bases, achieving

technical consistency across teams, managing stakeholders, balancing a shortage of expert resources, and aligning autonomous teams can become problematic [3, 4]. Practitioners of large-scale agile need to understand how to organize for scale, select optimal large-scale practices, and enable inter-team knowledge sharing [1, 5]. Development teams need to manage dependencies between, for example, requirements, testing, integration, and deliverables, working together with requirement engineers, architects, testers, other teams, and support and expert roles, all while keeping in line with the team's goals and prioritizations [3]. Many of these aspects represent coordination challenges, as several parts of the development organization depend on each other to align their efforts to deliver a software product.

Coordination is often defined as managing dependencies between activities [6], and effective coordination is considered a critical element for large-scale software development [5, 7]. Successful coordination is achieved by the use of appropriate coordination mechanisms, defined as organizational arrangements such as meetings, roles, tools, and artifacts associated with one or more dependencies that allow individuals or teams to realize collective performance [8]. When a set of coordination mechanisms are used to manage dependencies, it is known as a coordination strategy [9]. Moving to the inter-team level, coordination mechanisms, and potentially strategies, are directed at managing the dependencies between teams [2, 10]. Examples of mechanisms include Scrum-of-Scrum meetings and communities of practice, where representatives from each team are present [11], as well as tools and artifacts such as inter-team task boards and project backlogs. When mechanisms work together to address specific coordination issues, for instance managing inter-team prioritizations, they form coordination strategies.

While there is a vast literature on coordination in agile software development, research-based knowledge on inter-team coordination strategies is limited, as existing empirical studies have focused on coordination strategies within the team [9, 12]. To better understand the challenges to inter-team coordination and how they can be managed, we address the following research question: *How are coordination strategies used in large-scale agile to manage inter-team coordination challenges?*

We conducted a case study over six months in a large-scale program in the public transportation sector with 16 development teams. We analyzed data from interviews and field observations to identify the challenges. To guide our analysis, we applied concepts from the theory of coordination in co-located agile software development [9, 12], or the theory of coordination, for brevity [8]. This theory was developed in the context of co-located agile teams [9]. Of particular interest to further exploration is that the theory proposes one agile coordination strategy [9]. In large-scale contexts, there is likely to be a mix of typical agile software development practices and more traditional practices. Furthermore, as large-scale settings are characterized by complex dependencies [2, 3], there may be more than one strategy at play [10, 13].

2 Background and Related Work

2.1 Managing Dependencies in Large-scale Agile Development

Dependencies are central to the study of coordination. A dependency is defined as when the progress of one action relies upon the timely output of a previous action or on the presence of a specific thing, such as an artifact, a person, or relevant information [12]. Moving to the large-scale level, an inter-team dependency occurs when the output of one team is required as input for another team's work [2, 10]. According to a dependency taxonomy for agile projects [12], there are eight types of dependencies, divided into three categories: knowledge, process, and resource dependencies. Table 1 summarizes the eight types of dependencies.

Prior research suggests that there are many and complex dependencies in large-scale agile, and that organizational context matters for large-scale coordination. Uludağ and colleagues [14] studied recurring development patterns and presented an iteration dependency matrix to visualize dependencies between teams. Sekitoleko et al. [15] investigated challenges associated with communication of technical dependencies in large-scale agile. They found challenges such as planning, task prioritization, code quality, and integration and suggested that these challenges can be addressed by practices such as Scrum-of-Scrum meetings, continuous integration, and working in an open space [15]. Dingsøyr et al. [5] explored coordination in a large-scale program with a high degree of task uncertainty and interdependencies and highlighted the importance of scheduled and unscheduled meetings for coordination by feedback. They also emphasized the need for changing coordination practices over time [5].

Further, Gustavsson [16] studied coordination in companies that had implemented the Scaled Agile Framework (SAFe) and found that SAFe provides several coordination mechanisms, such as product increment planning meetings, Scrum-of-Scrum meetings, and program task boards address inter-team dependencies. These, however, required tailoring to the specific contexts of each company [16]. Martini et al. [17] also highlighted the role of context for coordination between teams. They studied inter-group interaction speed in an embedded software development context, exploring how boundary-spanning roles, activities, and artifacts mitigate challenges, with interaction hindering speed between teams. Their findings highlight the need for boundary-spanning mechanisms across teams and organizational levels for software architecture, processes, shared responsibilities, and managing expectations [17].

2.2 Coordination Strategies

One way to manage dependencies in software projects is to implement coordination strategies [9]. The idea that coordination mechanisms can be used together in the form of coordination strategies is not entirely new. Within software engineering, the concept has been explored conceptually in co-located [9, 12] and global software development settings [18]. However, empirical descriptions of the concept are scarce.

Xu [13] proposed eight coordination strategies for large agile projects for empirical exploration, focusing on decision-making, communication, and control as relevant dimensions of large-scale coordination and encouraging empirical exploration of these. Li and Maedche [18] conceptually explored coordination strategies within teams in a distributed setting,

suggesting that increased communication within the team facilitates shared understandings within the distributed team. Scheerer and colleagues [10] described eight types of inter-team coordination strategies, from purely mechanistic to cognitive and organic, and suggested that future research further explore the concept. These studies recognize that situational factors influence coordination strategies, which should also be relevant to the large-scale inter-team context, where teams are often surrounded by complex organizational contexts [19].

In this paper, we apply concepts developed in the theory of coordination [9, 12]. We chose this theory as a lens for investigating inter-team coordination because it provides a framework for analyzing dependencies and coordination mechanisms specific to agile software development and captures both explicit (such as a Kanban board) and implicit forms of coordination (such as shared knowledge) [5, 9]. The theory, and in particular the coordination strategy concept, is relevant also to large-scale contexts because it takes into account that project complexity and uncertainty, as well as the organizational structure, influence coordination [9]. The theory of coordination proposes that coordination in agile software development results from a combination of various agile coordination mechanisms, such as daily stand-up meetings, product backlogs, and software demos, which address dependencies in different ways [9, 12, 20]. The theory further proposes that appropriate coordination strategies enable effective coordination [20].

A coordination strategy comprises three components: coordination mechanisms for synchronization, for structure, and for boundary spanning [9, 20]. Synchronization activities and artifacts refer to coordination mechanisms that promote shared understanding. Structure coordination mechanisms include the proximity, availability, and substitutability of personnel, whereas boundary spanning refers to mechanisms that involve interaction outside the boundaries of the development team [9, 20].

Table 1. Types of Dependencies that can affect Agile Project Progress [24, 25]

Knowledge	A form of information is required for a project to progress	Requirement: Domain knowledge or a requirement is not known and must be located or identified.
		Expertise: Information about task is known only by certain persons or groups.
		Historical: Knowledge about past decisions is needed.
Process	A task must be completed before another task can process and this affects project progress	Task Allocation: Who is doing what, and when, is unknown.
		Activity: An activity is blocked until another activity is complete.
Resource	An object is required for a project to progress	Business process: Existing business processes cause a certain order of activities.
		Entity: A resource (person, place or thing) is not available.
		Technical: A technical aspect of development affects progress, such as when two software components must interact.

3 Method and Analysis

This study reports on a case study conducted in a Norwegian public sector organization. This organization has an ongoing development program, referred to as the PubTrans program. The data reported in this study was collected over six months during fall 2019. The case study design was chosen because the research-based knowledge on inter-team coordination of software development activities is limited, and case studies can provide detailed insights into the topic under investigation [21]. We took an ethnographic approach to the data collection, focusing on obtaining rich descriptions of the development process and the participants' experiences [22], complemented by in-depth interviews and document analyses.

3.1 Case Description

The PubTrans development program was established in 2016 following a public transportation reform and aims to develop a new micro-services-based platform. The new platform provides, among others, a sales platform for travel operators and a trip planner for travelers. Many languages and technologies were used across the program, and new technologies and tools were adopted as development needs arose. The new cloud-based platform ran on Google Cloud Platform with Kubernetes. Central languages and technologies in use included Kotlin and Java for back-end, and JavaScript (Node.js) and React-Native for front-end. They also used support tools such as Grafana, Prometheus, Slack, JIRA and Confluence. The development organization was mostly co-located with 16 teams, each responsible for their part of the overall software product.

Since the outset, PubTrans has worked with agile methods and autonomous teams. The agile values were largely embraced on the organizational level and the development management had top managements' support on working in agile ways of working. PubTrans did not subscribe to any specific agile methodology or large-scale agile framework, such as Scrum or SAFe. Rather, the development teams had the autonomy to choose which agile practices to use. Most teams had chosen to adopt practices from Scrum such as sprints, stand-up meetings and retrospectives with varying frequency. In addition to developers, all teams included a team leader, a tech lead (a form of team architect), and a product owner. In addition, there were several inter-team roles such as system-, cloud-, and security architects, as well as product and development managers.

Since the initial architecture and team organization was designed in 2016, PubTrans grew from five initial teams to the current large-scale set-up with 16 permanent teams. The teams were organized based on product areas, and the number of members per team varied from five to over fifteen team members. The program was initiated as a development project in 2016 but was transformed to an ongoing development program in 2018. Along the way, they went through several organizational phases and how to best align the team organization with the technical platform was an ongoing discussion.

While the new micro-services-based platform was being developed, PubTrans also delivered services both to their clients (typically public transportation operators) and to the general public through the old, monolithic system. More functionality was added to the new platform continuously and needed to be compatible also with the old system. Many

dependencies existed between these systems, and all teams had dependencies to other teams. In addition, there were inter-team knowledge and process dependencies related to, for instance, the delivery sequences. Accordingly, the need for coordination across teams was high.

3.2 Data Collection and Analytical Procedures

During fall 2019, we spent a total of 24 full days at the PubTrans site. The observations consisted of more than 44 hours of observation, including a total of 25 meetings. We conducted 12 interviews with team members and program managers. Additionally, we had frequent informal conversations with the program members. We also inspected documents, logs, and other textual sources for supplemental analysis. The data sources are specified in Table 2. All interviews were tape-recorded based on participants' consent and later transcribed by the first author. The duration was 62 minutes on average. All interviews were semi-structured, and although the conversations developed naturally, we used an interview guide with questions relating to participants' work habits and inter-team coordination practices. Questions included, *"What challenges do you face working with other teams or roles in the program?"*, *"Can you describe how you interact with members of other teams?"*, and *"What may hinder teams from completing their tasks?"*

When analyzing the data, we triangulated between sources to strengthen the accuracy and compellability of our findings [21]. By interviewing participants from different parts of the development organization, we gained access to participants' understanding of their work routines across teams and across levels of responsibility. By observing the development process as it unfolded over time and examining associated documents, we obtained context to the interview statements. Together, these data sources provided us with rich information for addressing our research question.

The data was coded in NVivo 12 by the first author, who knew the case in detail. To ensure validity, all emerging categories and concepts were negotiated during a series of discussions among the authors, and some of the material was coded by all authors before discussion. The analytical coding proceeded incrementally. During first-cycle coding, we used descriptive and holistic coding to understand "what is going on" in the data [23] and to identify the broad challenges observed and described by the participants. In the second stage, we categorized the challenges that were relevant across teams and identified the various dependencies and coordination mechanisms associated with inter-team challenges using focused coding [23]. Finally, we compared the challenges identified in the first stage with the dependencies and

Table 2. Data Sources

Data type	Description
Interviews	3 program architects, 3 tech leads, 1 product owner, 1 team leader, 4 program managers.
Observations	Twenty-four days on-site including observation of 6 tech lead forums, 6 stand-up meetings, 4 product owner meetings, 4 client meetings, 3 program demos, 2 retrospectives
Supplemental documents	Jira and Confluence documentation such as product backlog and prioritization documents, Slack channels; meeting agendas

related mechanisms. We considered something a coordination mechanism if it was associated with one or more distinct dependencies, and a coordination strategy when the mechanisms addressed the same set of challenges [9]. As the mechanisms included operated at the inter-team level, we considered them all to be boundary-spanning [9].

3.3 Limitations and Threats to Validity

All empirical studies have limitations that might threaten the validity and reliability of the results. One limitation of this study is the reliance on a single case. As such, the general criticisms of single-case studies, including the replicability and generalizability to other settings, apply to our study [21]. However, there is theoretical generalizability in the concepts applied, as the challenges we report on are not expected to be unique to this setting [21]. A second limitation relates to the reliance on interviews as a major data source. However, we complemented the interviews with extensive on-site observations and supplemental documents. As such, data triangulation allowed us to obtain context for the interview statements and strengthen our findings [21]. A third limitation is related to the number and types of meetings we observed. If we had observed more and different meetings, such as more retrospectives, we might have found other challenges and mechanisms. However, our extensive on-site presence allowed us to observe many of the challenges in practice.

4 Findings

In this section, we present four coordination strategies that were used to manage challenges with inter-team coordination in the large-scale program. Below, we describe the challenges, dependencies, and corresponding coordination strategies in more detail. The coordination strategies were: 1) aligning autonomous teams, 2) gaining and maintaining overview across teams, 3) managing prioritization issues, and 4) managing architecture and technical dependencies. Table 3 provides an overview.

4.1 Strategy 1: Aligning Autonomous Teams

One set of challenges was related to aligning autonomous teams in the large-scale program. Providing the teams with a high degree of autonomy resulted in process dependencies such as teams blocking each other, as well as the surrounding organizational business processes, which could cause delays that slowed down the speed of the program. Additionally, lack of alignment resulted in technical dependencies not being sufficiently managed. PubTrans aimed to facilitate an agile environment and culture based on autonomous teams. For instance, the teams could choose whether they wanted to apply Scrum, Kanban, Scrumban, or any other agile method. Although autonomy was appreciated, there were challenges related to the freedom of choice when teams operated with different definitions of done, had different testing regimes, and different ways of updating their documentation. One informant stated, *“Here, one has chosen a model with autonomous teams that are allowed to define their own ways of working. If there are sixteen teams here, there are sixteen different ways of doing things”* [Manager 4].

The missing alignment was also observed when we examined the teams’ Jira and Confluence pages; some had well-described processes and documentation, whereas others had

little to none. In addition, missing alignment contributed to a lack of technical consistency across teams. “We have allowed people to develop the new APIs team by team. That means they are not uniform” [Manager 2]. Although team autonomy was appreciated, teams also saw the need for alignment across teams: “It is great that the teams are free and have a lot of responsibility. But it is also essential to have arenas where we can discuss and share knowledge across teams so that it’s not spinning out of control” [Tech lead 2].

The challenges described above were addressed with several coordination mechanisms. PubTrans implemented shared documentation routines on Confluence, and shared delivery routines where a shared definition of done and common testing routines was central.

Table 3. Challenges and coordination mechanisms in the four strategies

	Challenge description	Coordination mechanisms
Strategy 1: Alignment	<p>Choice of agile methods result in different team routines:</p> <ul style="list-style-type: none"> - Different definitions of done - Different development routines - Different testing routines - Lack of technical consistency <p>Related dependencies: Process: Activity and Business process dependencies Resource: Technical dependencies</p>	<p>Synchronization activities: Inter-team stand-ups and status meetings, tech lead forum</p> <p>Synchronization tools and artefacts: Shared routines for deliveries and documentation and testing, common definition of done, test team, platform team to support teams with shared technologies</p> <p>Structure mechanisms: Co-location, open office space</p>
Strategy 2: Overview	<p>Large-scale makes it hard to maintain overview:</p> <ul style="list-style-type: none"> - Feeling out of sync with other teams - Problems with information flow - Task-related communication across teams - Locating people and information <p>Related Dependencies: Knowledge: Expertise, Task allocation, Requirement dependencies</p>	<p>Synchronization activities: Inter-team stand-ups and status meetings, program demo</p> <p>Synchronization tools and artefacts: Slack, shared backlog in Jira, organization map on Confluence, program roadmap, Objectives and Key Results</p> <p>Structure mechanisms: Open office space, co-location</p>
Strategy 3: Prioritization	<p>Hard deadlines and many clients lead to prioritization challenges:</p> <ul style="list-style-type: none"> - Stakeholder expectation management - Time and delivery pressure - Lack of time to prioritize quality work - Changing prioritizations - Lack of clarity in the prioritization process <p>Related Dependencies: Process: Activity dependencies Resource: Entity and technical dependencies</p>	<p>Synchronization activities: Inter-team stand-up meetings, Product owner meetings</p> <p>Synchronization tools and artefacts: Prioritization task board, shared backlog</p> <p>Structure mechanisms: Temporary team arrangements (task force teams, taking on other teams’ tasks)</p>
Strategy 4: Architecture	<p>Complex technical dependencies:</p> <ul style="list-style-type: none"> - Two systems in use in parallel - Teams becoming bottlenecks - Large code bases of some teams - Risk of repeating old patterns - Vulnerability for errors <p>Related Dependencies: Process: Activity dependencies, Resource: Technical dependencies</p>	<p>Synchronization activities: Tech lead forum</p> <p>Synchronization tools and artefacts: Objectives and Key Results, platform team</p> <p>Structure mechanisms: Temporary team arrangements</p>
<p>Note. Some coordination mechanisms are recurring across the strategies as they address more than one dependency.</p>		

Furthermore, they had established a platform team whose main responsibility was to support the development teams by *“developing functionality across teams, but also handling things like automatic builds, deploying, monitoring and logging overall across the teams”* [Team leader].

Other mechanisms included synchronization activities such as inter-team stand-ups for alignment of prioritizations, a test team that worked with testing across the teams, and a tech lead forum for addressing technical dependencies and architecture. Together, these mechanisms form a coordination strategy aiming to align the autonomous teams toward collective deliveries, while at the same time allowing the teams autonomy within appropriate boundaries.

Kn4.2 Strategy 2: Gaining and Maintaining Overview Across Teams

Another major challenge to inter-team coordination was the difficulty of maintaining overview across teams. In the interviews, participants described challenges such as being out of sync with other teams; problems with the information flow; locating information concerning other teams; and insufficient communication about tasks across teams. One informant explained, *“Right now, it is a bit hard to know the status of any given team. I don’t know where to find it. You need to play detective”* [Product owner]. Another said, *“It is an information problem. The technical state is not visible across teams and this is the greatest hindrance to addressing inter-team technical problems”* [Architect 2]. These challenges are examples of knowledge dependencies, such as expertise, task allocation, and requirement dependencies, because there is a need to know something about other teams in order to proceed on some action. In the team area, we observed expertise dependencies in practice when frustrated developers discussed whom they should talk to and who knew what in other teams.

The challenge with overview across teams was addressed by several coordination mechanisms. For instance, the office space supported overview and knowledge sharing by both providing open spaces for conducting inter-team stand-up meetings and supporting spontaneous informal coordination. A weekly program demo where the teams showcased their latest work was conducted in an open workspace (shown in Figure 1). In addition, a program roadmap was visible to all in the open work space. To help team members identify each other, the program had a Confluence document with the names and photos of all members of each of the 16 teams, as well as other employees in the program, such as managers and program architects.

Furthermore, Slack channels and direct messages provided the developers with an easy way of sharing knowledge and reaching out to people they did not know. PubTrans also used Objectives and Key Results (OKRs), which is goal-setting framework where objectives and corresponding key results are defined for individual teams and at the organizational level to measure progress over a set time period, typically per quarter [24]. The company used OKRs as a mechanism to provide an overview of the increasingly complex development process. OKRs were formed for all teams during off-site quarterly workshops where product owners, team leaders, and program architects and managers worked iteratively with forming team-specific objectives and key results. Because of the many inter-team dependencies, it was important to compare and discuss OKRs across teams and adjust as needed. *“The goal of using OKRs is to get an overview and gain insight in the organization. OKR allows us to work more structured, and gain overview of ‘this is where we are now’. Then we can assess what to*

focus on and use it to take action.” [Architect 1]. Together, these activities and artifacts form a coordination strategy for gaining and maintaining overview across teams.

4.3 Managing Prioritization Issues

Because PubTrans was started as a result of a political reform, there were often hard deadlines the program needed to adhere to, causing time and delivery pressure. One tech lead explained how this impacted the prioritizations they could make: *“Because of these deadlines we are forced to make very hard prioritizations. And that is something I’m sure the clients feel. It’s a bit painful from time to time”* [Tech lead 1]. PubTrans had many clients with different needs and the program sometimes overpromised what they were able to do. A manager explained, *“Things come up from different clients that they all expect us to solve. Sometimes we have not managed the expectations well enough, and we may simply not have finished on time”* [Manager 1]. Sometimes one team was forced to stop working on one task to prioritize something else with higher priority, which could cause delays for other teams. One tech lead illustrated a situation where three teams were working together: *“We were so close to finishing the feature! And then one of the teams had to prioritize something else”* [Tech lead 2].

Always chasing the nearest fixed deadlines had consequences for the overall product quality. Informants expressed the challenge of reducing technical debt and working on improvements: *“We need mechanisms that prevent us from always rejecting improvement work in favor of new features”* [Manager 3]. Another said, *“It’s all about not overloading the team and setting aside time to prioritize improvement”* [Tech lead 1].

There was also a lack of clarity in the prioritization process. The product owners were in charge of the functional prioritizations and were given input from four account managers who were responsible for client communication. Furthermore, clients could communicate directly with the teams through Slack. Although frequent communication with the customers was important, this set-up led to some confusion. One manager related this to the scaling of the program: *“In the beginning, everything was clear. But now, as things are expanding, these considerations of prioritizations start to matter. Who are in charge of what is going to be prioritized? Right now, sitting in this chair, I still do not know how our overall prioritization mechanism works”* [Manager 3].

The prioritization challenges relate to process dependencies, as they impacted task completion when an activity was dependent on input from several teams. They also relate to resource dependencies as often both technical features and input from members of other teams or program experts, such as the architects, were required to proceed.

Managing prioritization across teams was addressed with mechanisms such as temporary task force teams. A tech lead explained how they had successfully assembled a task force team to implement a feature where multiple teams were involved: *“To get things done as soon as possible, we put together members from four teams. We sat together and held our own task force stand-ups, focusing only on what we needed to get through”* [Tech lead 2]. Furthermore, teams taking on tasks from other teams was described as a successful mechanism when prioritizations caused delays across teams. One team readjusted by implementing a feature for a team that had too much on their hands instead of waiting for them to do it. *“The payment solutions were implemented fully by another team, which was a great success and one of the smarter things we have done”* [Product owner]. To manage the prioritization process, PubTrans used inter-team status meetings for product owners and team leaders, where they discussed top priorities from the different teams toward the overall deliveries. These were conducted in front of a physical task board showcasing the most important inter-team prioritizations. The product owners also had weekly meetings discussing the prioritizations in more detail. Finally, a new and refined shared backlog was created to help with prioritizing across teams and clients. Together, these mechanisms form a coordination strategy for managing inter-team prioritization.

4.4 Managing Architecture and Technical Dependencies

The program scaled fast, growing from five teams in 2016 to 16 teams in 2019. In addition, new clients were constantly added. Scaling up meant that new technical and architectural dependencies arose; several software components from different teams needed to interact, knowledge dependencies arose as information was required across teams, as well as process dependencies, because development activities had to be completed across teams before they were integrated.



Fig.2. The office space with the program’s roadmap easily visible

In developing the new micro-services-based application, it was hard to avoid developing copies of the old system, which, according to a team leader, left them at risk of developing a distributed monolith. *“Overall, we don’t have any mechanism to protect us from repeating old patterns. We have some teams that have been able to create something entirely new, but we also have teams that simply re-implement what they have implemented in the past”* [Team leader]. After some time, two teams who developed key components became bottlenecks. At one point, one team’s code base was seemingly large enough to constitute a mini version of the whole platform on which all teams depended. Furthermore, change in one part of the code could have a significant impact on other parts and make the platform vulnerable: *“One risk with developing a distributed monolith with poor error handling is that if one application goes down, the whole system goes down”* [Team leader].

Several coordination mechanisms were used to deal with these challenges. The architects formed specific OKRs to increase awareness of the technical state across teams and to identify constraints and bottlenecks that slowed down the delivery speed. The above-mentioned use of temporary team arrangements, the tech lead forum, as well as the platform team, also contributed to managing technical dependencies. Together, these coordination mechanisms form a coordination strategy for managing architecture and technical dependencies.

The tech lead forum was vital in this strategy because teams learned about each other’s architectures and discussed their challenges in relation to each other. The forum was described as a community of practice meeting aimed at sharing architecture-related knowledge and providing an overview of technical dependencies across all the teams. The forum met biweekly, facilitated by one of the program architects and accompanied by a Confluence page where meeting agendas and minutes were posted. One tech lead stated, *“I think the forum is great! It is very good to learn about other teams and what they do and what challenges they have. It’s very helpful”* [Tech lead 2].

While valuable, such synchronization activities introduced new challenges, as all teams needed to be represented. Challenges included keeping the meetings relevant for all, engaging participants in discussions, and finding the optimal meeting size. Across the six tech lead forums we observed, between 20 and 25 people showed up, of whom several were managers who were interested in following the discussions. Being a popular meeting, there was a shortage of space, and at two meetings, some people had to stand because there were no more chairs available. Despite the many participants, there were mostly five or six people who talked. One tech lead reflected on why people did not speak up: *“It can be very quiet in tech lead forum. Maybe it is that we do not dare to use the time of all these important people who are here”* [Tech lead 2]. A final challenge with this forum was related to its dependency on a person (entity dependency): *“The tech lead forum is currently completely dependent on the architect facilitating it; it is not self-organizing in any way”* [Team leader].

5 Discussion

In this study, we explored the research question: *How are coordination strategies used in large-scale agile to manage inter-team coordination challenges?* We applied the theory of coordination as a guiding lens and extended the coordination strategy concept to the inter-team

level. Our findings broaden the application of the theory of coordination beyond single co-located agile teams [9, 20] and answer calls for future research on coordination strategies [10, 13, 18].

According to the theory of coordination, a coordination strategy is a set of agile coordination mechanisms used to manage dependencies [9, 20]. This theoretical lens served to understand how PubTrans worked on solving their day-to-day coordination challenges among the 16 teams. These challenges are not unique to PubTrans, but rather are characteristics of scaling agile [4]. The four coordination strategies we identified from PubTrans' coordination challenges and mechanisms were 1) aligning autonomous teams, 2) maintaining overview in the large-scale setting, 3) managing prioritizations, and 4) managing architecture and technical dependencies. By extending the theory of coordination to the large-scale level, we show that the identified coordination strategies reflect the complex environment. In large-scale settings, agile practices are often used in combination with other organizational practices [2, 5]. We found that the four coordination strategies included both agile coordination practices, such as stand-up meetings, demos, and task-boards, as well as non-agile practices like OKRs, task force teams, and communities of practice.

Our findings further show that coordination mechanisms were used for several purposes to address challenges and dependencies in the program, which is reflected by their occurrence in several strategies. For instance, tools like Confluence and Slack supported both inter-team alignment (strategy 1) and overview of team members (strategy 2), by providing digital arenas for common documentation routines and gaining easy access to people. Inter-team stand-up meetings provided overview of what was going on in the teams (strategy 2) and served to manage prioritizations between teams (strategy 3). The use of temporary team arrangements supported both inter-team prioritizations (strategy 3), as well as technological dependency management (strategy 4).

Further, PubTrans had several coordinator roles [9, 25], such as the team leaders, product owners, and tech leads, as well as managers and program architects. Other studies highlight shared goals and knowledge enabled by high-quality communication between inter-team roles [3, 25, 26]. For instance, Sablis et al. [3] emphasize the importance of expert roles such as architects in supporting teams and that there is often a shortage of expertise in large-scale projects. In line with this research, we found that there were entity dependencies related to architects in facilitating the tech lead forum. Shastri and colleagues [26] found that project managers perform important coordinating activities such as facilitating, tracking, and negotiating project progress. This research relates to our findings in that program managers facilitated the use of OKRs and supported product owners and team leaders with inter-team prioritizations.

In large-scale software development, neither dependencies nor coordination needs are static. We found that the coordination strategies responded to coordination problems that emerged when the program scaled. Our findings are consistent with a study of two large-scale programs, where coordination mechanisms did not arise as ready-to-use procedures, but were formed during the coordination process [27].

5.1 Implications for Practice

Our findings generate a number of practical implications. While autonomous teams need to know what others are doing, solve technical dependencies, and align their prioritizations and processes with other teams [28], agile methods offer little specific advice on how this should be implemented in large-scale settings. In line with research on large-scale agile frameworks [16, 29] and hybrid settings [2], we found that coordination needs tailoring to the specific organizational context to cope with uncertainty, novelty, and complexity [6, 30]. Our results show that the coordination strategy concept is useful for dependency management at scale, and that large-scale agile programs benefit from adapting coordination mechanisms to their specific needs. We suggest that large-scale companies gather insights of their coordination challenges and dependencies across teams and use these to understand their own coordination strategies.

With respect to the first strategy, aligning autonomous teams, we find that while autonomous teams are central to agile, it appears important to strike a balance between autonomy and alignment and to be flexible across the large-scale development organization [2, 4, 31]. We suggest including shared documentation and testing routines and a common definition of done while still allowing the teams autonomy to choose development practices in an alignment strategy.

The second strategy, maintaining overview across teams, relates to typical challenges with knowledge dependencies as the number of teams grows so large that it is hard to keep track of who is working on what. For this strategy, we recommend including mechanisms such as keeping a team chart showcasing who does what in which teams, and using communication tools that provide easy access to members of other teams, such as Slack, and regular synchronization meetings to support overview [27].

Relating to the third strategy, managing prioritizations, PubTrans worked on establishing effective prioritization mechanisms. In line with previous studies [e.g., 5, 16, 27], we found that physical or digital prioritization boards highlight essential inter-team prioritizations and guided teams in adjusting to each other. Another successful practice in PubTrans was the ability of teams to take on the tasks of other teams. This flexibility appears core to an agile culture and mindset. We recommend such practices to make the most of a strategy for managing prioritizations. Concerning the fourth strategy, managing architecture and technical dependencies, we recommend the use of communities of practice, such as the tech lead forum, to support management of technical dependencies across teams [11], and establishing a platform team to support development teams [29].

6 Conclusion and Future Research

In this study, we explored the research question of how coordination strategies were used to manage challenges with inter-team coordination in a large-scale agile program with 16 teams. We found the coordination strategy concept useful for studying inter-team coordination in large-scale settings. The concept provides practitioners with an approach that is highly context-specific and flexible and thus suitable for the volatile, complex, and ambiguous large-scale development setting. From our analysis, we found four coordination strategies: 1) aligning autonomous, 2) gaining and maintaining overview across teams, 3) managing prioritization

issues and, 4) managing architecture and technical dependencies. We extend the coordination strategy concept to include more practices beyond agile coordination mechanisms, as we found that the mechanisms included in the strategies consisted of both agile practices, such as stand-up meetings and demos, and other practices such as OKRs and a community of practice. Future research could further explore how coordination mechanisms fit together to form coordination strategies, and how to tailor them to contribute to effective coordination in large-scale settings. We also encourage future research to explore coordinator roles in relation to inter-team coordination strategies. Finally, our on-site access allowed us to explore coordination in a co-located setting. Since then, the workplace has changed, and we encourage empirical research on coordination strategies in distributed settings.

Acknowledgements

This research was supported by the Research Council of Norway through the research project Autonomous teams (A-teams) project, under Grant Number 267704.

References

1. Bass, J.M., Salameh, A.: Agile at scale: a summary of the 8th International Workshop on Large-Scale Agile Development. Presented at the Agile Processes in Software Engineering and Extreme Programming–Workshops (2020).
2. Bick, S., Spohrer, K., Hoda, R., Scheerer, A., Heinzl, A.: Coordination challenges in large-scale software development: a case study of planning misalignment in hybrid settings. *IEEE Transactions on Software Engineering*. 44, 932–950 (2018).
3. Sablis, A., Smite, D., Moe, N.: Team-external coordination in large-scale software development projects. *Journal of Software: Evolution and Process*. e2297 (2020).
4. Dikert, K., Paasivaara, M., Lassenius, C.: Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*. 119, 87–108 (2016).
5. Dingsøy, T., Moe, N.B., Seim, E.A.: Coordinating Knowledge Work in Multi-Team Programs: Findings from a Large-Scale Agile Development Program. *Project Management Journal*. 49, 64–77 (2018).
6. Malone, T.W., Crowston, K.: The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*. 26, 87–119 (1994).
7. Faraj, S., Sproull, L.: Coordinating expertise in software development teams. *Management science*. 46, 1554–1568 (2000).
8. Okhuysen, G.A., Bechky, B.A.: 10 coordination in organizations: An integrative perspective. *Academy of Management annals*. 3, 463–502 (2009).
9. Strode, D.E., Huff, S.L., Hope, B., Link, S.: Coordination in co-located agile software development projects. *Journal of Systems and Software*. 85, 1222–1238 (2012).
10. Scheerer, A., Hildenbrand, T., Kude, T.: Coordination in large-scale agile software development: A multiteam systems perspective. Presented at the 2014 47th Hawaii international conference on system sciences (2014).
11. Smite, D., Moe, N.B., Levinta, G., Floryan, M.: Spotify Guilds: How to Succeed With Knowledge Sharing in Large-Scale Agile Organizations. *IEEE Software*. 36, 51–57 (2019).

12. Strode, D.E.: A dependency taxonomy for agile software development projects. *Information Systems Frontiers*. 18, 23–46 (2016).
13. Xu, P.: Coordination in large agile projects. *Review of Business Information Systems (RBIS)*. 13, (2009).
14. Uludağ, Ö., Harders, N.-M., Matthes, F.: Documenting recurring concerns and patterns in large-scale agile development. Presented at the Proceedings of the 24th European Conference on Pattern Languages of Programs (2019).
15. Sekitoleko, N., Evbota, F., Knauss, E., Sandberg, A., Chaudron, M., Olsson, H.H.: Technical dependency challenges in large-scale agile software development. Presented at the International conference on agile software development (2014).
16. Gustavsson, T.: Dynamics of Inter-Team Coordination Routines in Large-Scale Agile Software Development. In: Proceedings of the 27th European Conference on Information Systems (ECIS). pp. 1–16. , Uppsala (2019).
17. Martini, A., Pareto, L., Bosch, J.: A multiple case study on the inter-group interaction speed in large, embedded software companies employing agile. *Journal of Software: Evolution and Process*. 28, 4–26 (2016).
18. Li, Y., Maedche, A.: Formulating effective coordination strategies in agile global software development teams. (2012).
19. Mikalsen, M., Næsje, M., Reime, E.A., Solem, A.: Agile Autonomous Teams in Complex Organizations. Presented at the XP Workshops (2019).
20. Kanaparan, G., Strode, D.: A Theory of Coordination: From Propositions to Hypotheses in Agile Software Development. Presented at the Proceedings of the 54th Hawaii International Conference on System Sciences (2021).
21. Yin, R.K.: Case study research and applications: Design and methods. Sage publications (2018).
22. Sharp, H., Dittrich, Y., C. R. B. de Souza: The Role of Ethnographic Studies in Empirical Software Engineering. *IEEE Transactions on Software Engineering*. 42, 786–804 (2016).
23. Saldaña, J.: The coding manual for qualitative researchers. Sage (2012).
24. Niven, P.R., Lamorte, B.: Objectives and key results: Driving focus, alignment, and engagement with OKRs. John Wiley & Sons (2016).
25. Berntzen, M., Moe, N.B., Stray, V.: The product owner in large-scale agile: an empirical study through the lens of relational coordination theory. Presented at the International Conference on Agile Software Development (2019).
26. Shastri, Y., Hoda, R., Amor, R.: The role of the project manager in agile software development projects. *Journal of Systems and Software*. 173, 110871 (2021).
27. Moe, N.B., Dingsøy, T., Rolland, K.: To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software development. (2018).
28. Martini, A., Stray, V., Moe, N.B.: Technical-, social-and process debt in large-scale agile: an exploratory case-study. Presented at the International Conference on Agile Software Development (2019).
29. Paasivaara, M.: Adopting SAFe to scale agile in a globally distributed organization. Presented at the 2017 IEEE 12th International Conference on Global Software Engineering (ICGSE) (2017).
30. Jarzabkowski, P.A., Lê, J.K., Feldman, M.S.: Toward a Theory of Coordinating: Creating Coordinating Mechanisms in Practice. *Organization Science*. 23, 907–927 (2012).
31. Moe, N.B., Smite, D., Paasivaara, M., Lassenius, C.: Finding the Sweet Spot for Organizational Control and Team Autonomy in Large-Scale Agile Software Development. *Empirical Software Engineering*. (2021).

Paper 3: A Taxonomy of Inter-team Coordination Mechanisms in Large-Scale Agile

Marthe Berntzen, Rashina Hoda, Nils Brede Moe, and Viktoria Stray

IEEE Transactions on Software Engineering, 49, 2, (pp. 699-718), 2022

Abstract

In large-scale agile software development, many teams work together to achieve overarching project goals. The more teams, the greater the coordination requirements. Despite the growing popularity of large-scale agile, inter-team coordination is challenging to practice and research. We conducted a case study over 1.5 years in a large-scale software development firm to better understand which inter-team coordination mechanisms are used in large-scale agile and how they support inter-team coordination. Based on a thematic analysis of 31 interviews, 113 hours of observations, and supplemental material, we identified 27 inter-team coordination mechanisms. From this, we offer the following contributions. First, we propose a taxonomy of inter-team coordination with three categories: coordination meetings, such as communities of practice, inter-team stand-ups, and retrospectives; coordination roles, such as the program architects and the platform team; and coordination tools and artefacts, such as Slack and JIRA as well as inter-team task boards, product backlogs, and roadmaps. Second, the coordination mechanisms displayed combinations of four key characteristics, technical, organizational, physical, and social (TOPS), which form the basis of the TOPS framework to capture the multifaceted characteristics of coordination mechanisms. *Technical* relates to the software product and/or technical tools supporting software development. *Organizational* pertains to the structural aspects of the organization. *Physical* refers to tangible or spatial characteristics. *Social* captures interpersonal and community-based characteristics. Finally, the taxonomy and the TOPS framework provide a knowledge base and a structured approach for researchers to study as well as for software practitioners to understand and improve inter-team coordination in large-scale agile.

Index Terms—Large-scale agile, agile software development, inter-team coordination, case study, taxonomy

1 Introduction

WHEN developing software on a large scale, multiple teams work together over an extended period to realize shared development goals. To support the development process, agile practices are popular in large-scale settings. However, conducting successful large-scale agile software development is challenging [1]–[4]. Resistance or lack of commitment to agile practices, ensuring management support in agile ways of working, balancing the need for alignment with autonomy [2], [5], communication issues during requirements engineering and quality assurance [1], and planning misalignment between the team and inter-team levels [6], [7] are among the identified threats to large-scale agile [1].

Among these, coordination, or managing dependencies between activities [8], has been identified as a critical challenge [1], [2], [4], [6], [9]. In large projects, many forms of dependencies can lead to coordination challenges. Dependencies between tasks or activities constrain how and when each task can be performed [8], [10]–[12]. In large-scale agile, dependencies can be related to, for example, features and tasks, code, architecture, autonomous teams, expertise personnel, and on-site customer [5], [6], [13], [14]. Successful coordination of activities such as iteration and sprint planning, backlog grooming [15], bottom-up architecture design, product demonstrations, and continuous deployment and delivery [5], [16] dictate the success or failure of large-scale agile software development. Therefore, dependencies must be managed continuously throughout the development life cycle.

Coordination mechanisms are organizational processes, entities, and arrangements used to manage dependencies to realize a collective performance [8], [17], [18]. In large-scale agile, mechanisms are used to enable coordination within each development team, as well at the inter-team level. The latter is the focus of this study. Inter-team coordination mechanisms include agile meetings (e.g., stand-up and retrospective meetings) performed at the inter-team level [19], digital communication tools [20], inter-team groups such as communities of practice [3], [21]–[23] and specialized boundary-spanning roles [24] such as architects and product owners [25]. Although individual studies have described inter-team coordination mechanisms, there is no comprehensive collection of inter-team coordination mechanisms with an in-depth description of their categories and characteristics to guide large-scale agile coordination. As such, while we know much about various coordination mechanisms, systematic tools for identifying and evaluating mechanisms to guide research and practice are lacking. With this study, we contribute to filling this gap by addressing the following research question:

Which inter-team coordination mechanisms are used in large-scale agile software development and how do these mechanisms support inter-team coordination?

We conducted a case study in Entur, a public sector, large-scale development firm recognized as a successful and mature agile program within its national context. This ongoing development program has a complex product and many dependencies across teams, which made it a suitable case for studying inter-team coordination. A relatively long frame of reference and extensive access helped us gain a native, in-depth understanding of the coordination mechanisms used to address the inherent challenges of large-scale software development. Details of the case study are described in Section 3.

We used data collected from 31 interviews, 113 hours of observations, and supplemental material such as program documentation and communication logs from Slack to address our

research question. Based on thematic analysis [26], [27], in Section 4, we present 27 inter-team coordination mechanisms that form the empirical basis for a proposed taxonomy of inter-team coordination mechanisms under three categories:

- **Meetings**, such as inter-team stand-ups, communities of practice, and retrospectives.
- **Roles**, such as the method specialist and program architects.
- **Tools and artefacts**, such as Slack and Confluence, and inter-team task boards, product backlogs, and roadmaps.

Additionally, the study's in-depth nature enabled us to gather detailed characteristics and nuances of these mechanisms. We identified four key characteristics of inter-team coordination mechanisms:

- **Technical**, that is, the product- or software development-based,
- **Organizational**, the team and company structure based,
- **Physical**, the tangible characteristics, and
- **Social**, the inter-personal or community based.

Abbreviated as TOPS, these characteristics combine to form a novel framework. Finally, we develop a visual template (provided in Section 5) to demonstrate how the taxonomy and framework can be used in practice to analyze coordination mechanisms. The template provides an actionable approach for practitioners to assess and improve their inter-team coordination practices.

2 Background and Related Work

In this section, we present relevant background literature on large-scale agile software development, coordination mechanisms, and coordination challenges in large-scale agile. Finally, we introduce the need for a taxonomy of inter-team coordination mechanisms, which is further developed in the results and discussion.

2.1 Agile Software Development at Scale

The term *agile* refers to iterative and incremental approaches to software development based on an “agile philosophy” that centers around the core principles of valuing “*individuals and interactions over processes and tools*,” “*close collaboration with customers over contract negotiation*,” “*working software over comprehensive documentation*,” and “*responsiveness to change over following a plan*” [28]. As such, agile is not an out-of-the-box process or tool, but rather an umbrella term for methods and ways of working with software development based on agile values and principles [1].

In recent years, the popularity of agile has expanded well beyond small-team projects to the extent that today it seems as though almost every organizational process has the potential to “become agile” [29]. Although agile methods were originally intended for smaller projects [30] and primarily have been successful in small teams, agile principles and techniques are popular also in large-scale software development [29]. According to the latest State of Agile report, almost 70% of the survey respondents were employed in software development organizations with more than 100 individuals [31].

There is no single definition in the literature of what constitutes large-scale agile [2]. Although there is some agreement on the scale that qualifies it as large-scale (i.e., projects with more than six teams or involving more than 50 developers [1]), there is no agreement on a specific set of development methods or practices that constitute large-scale agile [2] or which large-scale practices are better [9].

A key characteristic of large-scale software development is the need to balance agile with the need for organizational-level alignment [2], [6], [7], [9]. A common approach is to use agile methods and tools at the team level and to use a hybrid of agile and traditional project management approaches at the inter-team level [2], [5], [14]. For example, an agile project might use retrospectives for team leaders (an agile team practice) but involve project managers and key performance indicators (a non-agile role and performance metric, respectively) as well. As such, the term “large-scale agile” does not refer to any specific set of methods, but represents a mix of agile and traditional tools and practices [2], [6], [7].

2.2 Perspectives on Coordination and Coordination Mechanisms

Researchers from a range of academic disciplines have studied coordination for decades. In organizational and management science, early contributions include Van de Ven et al.’s [32] coordination modes, and Thompson’s [33] notion of coordination by mutual adjustment, both representing explicit forms of coordination [34]. Later developments also take into account the dynamic and changing nature of coordination [17], [35]. Other approaches focus on the role of relationships in driving coordination through shared goals and knowledge and high-quality communication [36]. In teamwork studies and organizational psychology, implicit coordination has been studied from the perspectives of shared cognition [37], transactive memory systems [38], and shared mental models [39]. A detailed review of the literature on coordination in organizations can be found in [17]. Common to perspectives on coordination is the notion that interdependent tasks and activities are managed by the use of coordination mechanisms.

Many software engineering researchers adopt Malone and Crowston’s basic definition of coordination as the management of interdependent activities [8]. In their coordination theory, dependencies stem from shared resources, tasks, producer–consumer relationships, and simultaneity constraints. They do not provide a firm operationalization of coordination mechanisms, but provide examples of mechanisms such as scheduling, tracking, inventory management, and goal selection [11].

Attempts have been made to develop coordination mechanisms further into a more actionable concept. Okhuysen and Bechky [17, p. 472] defined coordination mechanisms as “organizational processes and arrangements that allow individuals to realize a collective performance.” This conceptualization makes sense in the large-scale agile setting where ongoing processes to manage dependencies between teams are key to successful software development. Schmidt and Simone [18] focus on the construction of coordination mechanisms in cooperative settings. They define coordination mechanisms as organizational constructs consisting of protocols, conventions, and procedures that are related to artifacts used to reduce the complexity of work [18], [40].

Researchers have argued for a more comprehensive framework to understand and describe coordination in relation to the software development process and the daily activities of software

engineers [41], [42]. Because large-scale agile consists of complex technical, organizational, and social processes taking place both digitally and physically, we believe a broader definition of coordination mechanisms is necessary to include a wider range of categories relevant to the large-scale agile setting.

In this study, we base our understanding of coordination on Malone and Crowston's basic definition [8], combined the view of coordination mechanisms as processes and arrangements [17], while recognizing the importance of artefacts, standards, protocols and similar entities [18]. From this, we define coordination mechanisms as organizational processes, entities, or arrangements, used to manage dependencies between activities, to realize a collective performance.

A coordination mechanism can be used for several purposes, and it must address at least one dependency [10], [12], [13]. Dependencies occur when the completion of a task or an action relies either on the output of a previous task or action, or the presence of some artefact, person, or information [13]. Examples of coordination mechanisms applied at the individual team level include product backlogs and wall boards [24], daily stand-up meetings [19], team-level specifications, wireframes [43], pair programming, and team-level domain specialists [13], to name a few. Strode [13] developed a dependency taxonomy for agile teams with *three categories* and *eight sub-categories*:

- **Knowledge dependencies** refers to information required for an individual or a team to proceed and it is comprised of *requirement*, *expertise*, *historical*, and *task-allocation* dependencies.
- **Process dependencies** refer to the order in which developmental or organizational tasks and activities must be completed and it consists of *activity* and *business process* dependencies.
- **Resource dependencies** refers to the need for specific objects, including an *entity* (a person, place, or thing), and *technical* dependencies, including software and architectural components.

Various coordination mechanisms are used to manage these dependencies. For instance, knowledge dependencies can be managed by stand-up meetings or product backlogs, process dependencies by burn down charts, and resource dependencies by “done” checklists and informal team communication [13]. Developed from research conducted within agile teams, this taxonomy provides an approach to coordination specific to agile development. Moving to the inter-team level calls for a further exploration of coordination mechanisms used for coordination between teams in large-scale agile.

2.3 Coordination Challenges in Large-Scale Agile

As the popularity of agile methods continues to grow, several challenges remain barriers to the success of large-scale agile. The notion of autonomous teams lies at the core of agile software development [30], [44]. However, in large-scale agile, team autonomy must be balanced with the larger organizational structures because of a greater need for coordination and alignment between the system, the organization, and the product [1], [6], [45]. Product complexity and technical dependencies may further require careful management in large systems, in particular those involving tightly coupled teams and architectures [5], [46]. These and other challenges,

such as coordinating between teams, managing stakeholders, and keeping to the agile principles, seem to prevent the success of large-scale agile [1], [2], [5]. Among these, inter-team coordination has been identified as a major challenge [1].

Inter-team coordination refers to coordination happening outside an individual team's boundaries, either with other teams or with roles operating between teams such as architects and agile coaches [47]. In complex, large-scale settings, ensuring optimal levels of inter-team coordination is far from straightforward as more teams, roles, and technologies are introduced across teams. Inter-team coordination problems may stem from a lack of shared knowledge about goals and prioritizations as well as inefficient communication [25], [48] and insufficient management of dependencies across teams [8], [10], [12].

In the face of such challenges, scaling frameworks attract practitioners' attention, such as the Spotify model [2], Large-Scale Scrum (LeSS) [49], and the Scaled Agile Framework (SAFe) [50]. Most large-scale frameworks propose mechanisms to handle dependencies arising in the development process [3]. In LeSS, for instance, Scrum activities such as sprint planning and backlog refinement are aggregated to the inter-team level [49]. In SAFe, the most widely used scaling framework [31], coordination mechanisms include specialist and expert roles such as *architects* to manage technical dependencies across teams and provide expert support as well as the so-called *agile release train* to coordinate product delivery across teams [51].

Additionally, many organizations, including our case organization, use a hybrid of methods or their own internal scaling methods [6], [25], [31]. A recent systematic literature review on large-scale agile showed that of 191 primary studies on 134 large-scale organizations, 49 organizations used a standard large-scale framework, such as SAFe, while a total of 85 organizations had adapted and tailored their approach to agile software development [2]. As empirical research on using large-scale frameworks develops, a key finding is that context-based agile tailoring is vital to capture and address each organization's unique coordination context [2], [52] as well as changes in coordination needs over time [35], [48]. Regardless of framework or approach, researchers and practitioners agree that coordination is key to the success of large-scale agile development.

2.4. Inter-team Coordination Mechanisms in Large-scale Agile

Software development is a complex activity, and the larger the project, the more dependencies there are likely to be because most development work is conducted in parallel by several teams [47]. In the large-scale agile context, dependencies constrain action across teams, requiring inter-team coordination. In these situations, using inter-team coordination mechanisms is a way to manage these dependencies. These mechanisms are similar to team-level mechanisms, such as task boards and stand-up meetings, but adapted for use at the inter-team level.

A central characteristic of large-scale software development is that agile tools and practices are often used alongside other approaches to project and organization management [6], [7], [48]. Previous research has shown that the need for more and different forms of coordination is central to large-scale projects compared to smaller agile projects [5], [16]. Large-scale agile requires more communication arenas, extensive use of digital communication tools [20], boundary-spanning coordinator roles such as project managers [53], and expert roles operating at the inter-team level, such as project or program architects [47], [51].

Previous research on large-scale agile development practices has identified and described several individual inter-team coordination mechanisms. Examples include planned and unplanned meetings [15], [47], [48], communication platforms and tools such as Slack and JIRA [20], groups of representatives (often referred to as communities of practice) [3], [21]–[23] boundary-spanner roles such as product owners and architects [24], [54], and open spaces for inter-team coordination [5]. We revisit existing research on inter-team coordination mechanisms in Section 5.

While studies recognize that coordination mechanisms can be used for several purposes [14], [35], [48], research has yet to examine the underlying categories and characteristics of coordination mechanisms in large-scale agile. Large-scale software development is a complex socio-technical activity, where several possible solutions to development problems are possible [55]. As such, there are many ways to design and implement technical software systems, some better than others. The same applies to the social organization of software projects or programs, which is arguably the reason agile approaches are popular today. This relates to an idea shared with the seminal literature on coordination, namely that there is no one best way to organize for optimal coordination [8]. Different coordination mechanisms may be used to manage dependencies in more or less efficient ways, depending on the situation [35]. Therefore, it made sense to approach our study from the basis of understanding both agile software development and inter-team coordination as socio-technical activities.

Although previous research has identified and described several individual coordination mechanisms used in large-scale agile, there is no collection or categorization of inter-team coordination mechanisms. As such, while there exist several accounts of individual coordination mechanisms, tools for identifying and evaluating mechanisms are lacking. Such tools would benefit both researchers in structuring the further study of inter-team coordination and practitioners in selecting appropriate mechanisms to manage their specific dependencies. With this study, we seek to begin this work by developing a taxonomy of inter-team coordination mechanisms in large-scale agile.

Taxonomies provide ways of systematically organizing knowledge in a domain of interest to allow the identification of a class of phenomena, and to compare and contradict classes [56]. Taxonomies are used to describe novel topics where concepts need to be identified, and when much is known about a topic, but that knowledge is yet to be meaningfully organized [56], [57]. They are useful in mapping knowledge gaps, directing future research within a field or topic of research and serving as basis for later development of process theories [56]. Within software engineering, examples include taxonomies for large-scale agile projects [58], software testing skills [59], and global software engineering [60]. To assess their appropriateness and relevance, taxonomies should be evaluated against predetermined quality criteria. We return to this in Section 5.

3 Research Design

In this section, we present details of our case organization, the data collection, and analytical procedures. We conducted a case study in a large-scale public sector IT organization in Norway. We chose a case study approach because we wanted to gain a deep understanding of

coordination mechanisms within a real-life context. Case studies are suitable to answer research questions requiring substantial depth and level of detail, in particular when the boundaries between the topic of study and its context is not clear [61], [62], such as the complex socio-technical activities involved in the coordination of large-scale software development. Our access to the case over 1.5 years provided ample opportunity to study the topic in depth. In our case study, we applied an ethnographic approach to the data collection procedures and a thematic analysis approach to the data analysis. Our presentation of the findings follows a style common to reporting the findings of similar case studies in software engineering, e.g. [6], [63]. Details on the data collection and analysis are presented in sections 3.2 and 3.3.

3.1 Case Description

Our case company, Entur, is a public sector IT organization established in 2016, following a public transportation reform initiated by the Norwegian Ministry of Public Transportation. We chose this case because it is an ongoing development program with a complex product and many dependencies across teams, making it an interesting case for studying inter-team coordination. The case has been regarded as a successful large-scale public software development program by the Digitization Council of Norway, a professionally independent body appointed by the Ministry of Local Government and Modernization (<https://www.digdir.no/digdir/about-norwegian-digitalisation-agency/887>). The program is further recognized as a mature agile program by practitioners within their national context.

Access was arranged through the third and fourth authors, who were first connected to the organization in 2017 through a funded research project. It became clear during this initial contact that this case represented a unique opportunity to study coordination in a fast-growing, large-scale agile company with a complex external environment and a diverse stakeholder group, stretching from end users of the product to governmental departments.

When the opportunity arose to conduct a case study in early fall 2018, three of the four authors met with Entur representatives to set up arrangements. During these initial meetings, we learned more about the organization, the team organization, and their challenging areas. Following these meetings, the first author commenced the data collection from August 2018 through January 2020.

3.1.1 Case Context

Entur's main goal is to develop and maintain a digital platform for public transportation in response to a political reform. Thus far, they have been successful in meeting the reform goals. Some of Entur's services include a travel planning application and online as well as physical systems for selling and distributing tickets. Its customers and users include public transportation operators in Norway that use its APIs and sales systems as well as individual travelers using the platform and its services. A vital part of the transportation reform was onboarding new transportation operators on Entur's platform and continuously developing the relationship with these operators. Therefore, Entur frequently held workshops, retrospectives, and meetings, and participated in a change advisory board with the major customers.

While the new platform was under development, the old system was maintained. The new cloud-based platform is built on modern architectural principles and is based on microservices,

whereas the old system has a monolithic structure. The new platform runs on Google Cloud Platform with Kubernetes and Firebase. During the course of our data collection, Entur was still dependent on the old system to provide its services, but the company was working towards making it redundant. Many languages and tools were used to develop the new platform, and Entur adopted new technologies as needed. Some central languages included Kotlin, Java, and Scala for back-end, and JavaScript (Node.js) and React-Native for front-end. Additionally, they used support tools such as Grafana, Prometheus, JIRA, Confluence, and Slack.

The relatively complex internal and external environment surrounding the development program led to a range of dependencies across teams. Examples of dependencies include technical dependencies between the old and new software platforms and between the development teams as well as knowledge dependencies due to a shortage of expert resources and the distribution of knowledge between teams. Process dependencies also resulted from autonomous teams with different development routines as well as from the surrounding organization. We return to these and other dependencies in Section 4.

3.1.2 The Large-Scale Agile Environment

Entur has worked with agile methods since it was established in 2016. The company does not subscribe to any specific large-scale framework but uses a hybrid of methods and practices based on the current development needs. Practices were subject to change as the organization scaled and new needs arose. From August 2018 to January 2020, the number of development teams grew from 13 to 17, and the number continued to grow after we concluded our data collection. As such, the use of agile practices in the program was not static but changed over time.

Overall, the teams had the autonomy to choose how to organize themselves and which agile practices, tools, and techniques to use in solving their team-specific development goals. Practices from Scrum and Kanban, such as stand-ups, retrospectives, product backlogs, and visual task boards, were commonly used. An important factor for the use of agile methods in the program was the support of top management and the board of directors to work in this way. Another was their ability to test and experiment with their ways of working to respond to their internal and external environment while simultaneously keeping up to speed delivering services to their clients and the public. This meant some practices emerged as the program scaled, whereas others disappeared. This ability to sense and respond was one of the large-scale agile program's strengths.

Entur organized its developers into teams that each had areas of responsibility towards the overall product. On average, the teams spent 40% of their time developing new features and 60% on maintenance, bug fixing, and improving the code (i.e., reducing technical debt). Each development team had a team leader, product owner, tech lead, and developers. Some team leaders and product owners were responsible for more than one team. The number of members per team ranged from five to 17. In addition to the team roles, there were roles at the inter-team level, such as program managers and architects, as well as customer managers (see Table 4). The teams worked in an open office landscape that was also used for open space sessions as well as for displaying inter-team tools and artefacts (see Section 4 for more details).

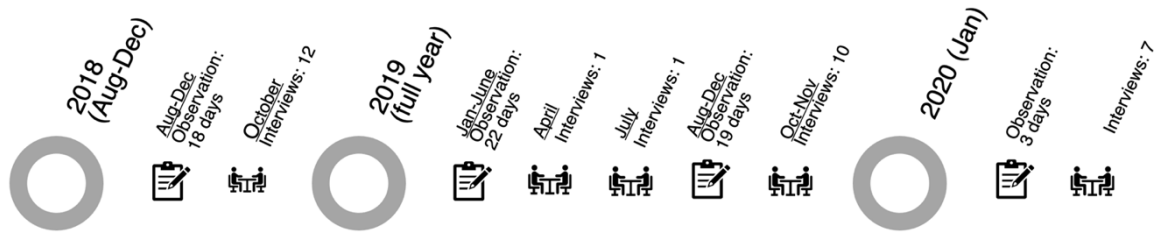


Fig. 1. Overview of the data collection from August 2018 to January 2020

Although keeping an agile mindset and providing the teams with the autonomy to self-organize was considered a strength in the program, its size and complexity also led to coordination challenges that warranted the need for shared routines and mechanisms across teams. During the course of our study, several such initiatives were taken, including using inter-team backlogs and prioritization documents, establishing more inter-team meetings such as communities of practice, and hiring a delivery process specialist responsible for implementing a shared delivery process that would better support future continuous integration and DevOps initiatives. These and other practices to be further described in Section 4 and Section 5.1.1., supported the program in balancing autonomy and alignment in the large-scale environment.

3.2 Data Collection

Our data consist of 113 hours of meeting observation across 62 days on site, 31 in-depth interviews, and a range of supplemental documentation. Data were collected from August 2018 through January 2020. Figure 1 provides an overview of main events during the data collection

TABLE 1
DATA COLLECTION DETAILS

MEETING OBSERVATIONS	113 hours of observation across 62 days on-site, including: <ul style="list-style-type: none"> • 10 prioritization meetings • 7 tech lead forums • 7 program demos • 6 product owner meetings • 6 inter-team stand-up meetings • 4 inter-team retrospectives • 2 OKR workshops • 26 ad hoc inter-team meetings • 26 intra-team meetings
INTERVIEWS	31 interviews with 25 participants (mean length 51 minutes). Participants included: <ul style="list-style-type: none"> • 10 product owners (6 male, mean IT tenure 11.5 years, mean company tenure 1.8 years) • 5 program managers (4 male, mean IT tenure 18 years, mean company tenure 1.6 years) • 4 program architects (4 male, mean IT tenure 19 years, mean company tenure 1.4 years) • 4 tech leads (3 male, mean IT tenure 7 years, mean company tenure 2.4 years) • 2 team leaders (2 male mean IT tenure 9 years, mean company tenure 1.5 years)
DOCUMENTATION	Slack logs, Confluence documentation, e-mails, internal and external company documents (e.g., presentations, reports)

period. Within our overarching case study approach, we collected data using a variety of sources and techniques, including interviews, project documentation and chats, and an ethnographic approach to the data collection [64]. We chose ethnographic data collection procedures such as participative observation and detailed note-taking as data collection mechanisms because it suited our aims of understanding people’s practices as they unfold in a natural setting [65]. Ethnographic approaches to data collection are typically defined by researcher immersion in the context of the participants and it traditionally involves long-term fieldwork where the researcher spends considerable time with the research participants, observing and documenting their everyday situations [64], [65]. Within software engineering, an ethnographic approach to data collection can “provide an in-depth understanding of the socio-technical realities surrounding everyday software development practice” [64, p. 786]. We considered this appropriate to our overall research question due to the opportunities for deep understanding provided.

Another defining characteristic of an ethnographic approach to data collection is extensive notetaking. During our time on-site, field notes were written following an observation protocol specifying the contents of the record, participants present, description of activities, direct quotes, snippets of conversations, researcher reflections on the observations, and any follow-up questions or concerns [65]. Notes were jotted down during meetings and observations and were refined at the end of each observation day. The field notes correspond to 216 pages of text (with standard MS Word margins, 11-point Calibri font). The in-depth descriptions resulting from the fieldwork, combined with the extensive field notes, resulted in a large and diverse data material that allowed for a detailed analysis. Table 1 provides an overview of the data, and the following sections provide more details.

Observations. The first author conducted the observations on an even basis throughout the data collection period (see Figure 1). We observed inter-team meetings where all teams were represented, including inter-team stand-ups and retrospectives, tech lead forums, and program demos. In addition, we observed ad hoc inter-team meetings where two or more teams were represented. We also observed intra-team meetings within the development teams. The intra-team meetings almost always covered inter-team aspects, which made them relevant to our analyses. In addition to the meetings, we also observed the development teams’ everyday work practices and engaged

Meeting observation

Raw data: Team leader retrospective.

Date: Tuesday, October 9, 2018, 8:25 AM.

Place: [Entur site, large meeting room with whiteboard and whiteboard pens.]

Participants: 13 participants, excluding the researcher. Present were team leaders and representatives of the teams, retrospective facilitator, a project manager, and development manager.

“We are sitting in a large meeting room downstairs, people sitting around a table. On one side of the room is a whiteboard, some sitting with their backs to it, but they can easily turn the chairs to see the board. The team leaders have stand-ups every Monday, and occasionally (the facilitator told me last time was during this summer) they have retrospectives focusing on inter-team collaboration from the team leader perspective. [...] Therefore, they did not have an ordinary stand-up this week.”

Code1: Inter-team meeting
Code2: Team leader retrospective
Code3: Physical set-up

Fig. 2. Field notes extract of a meeting observation

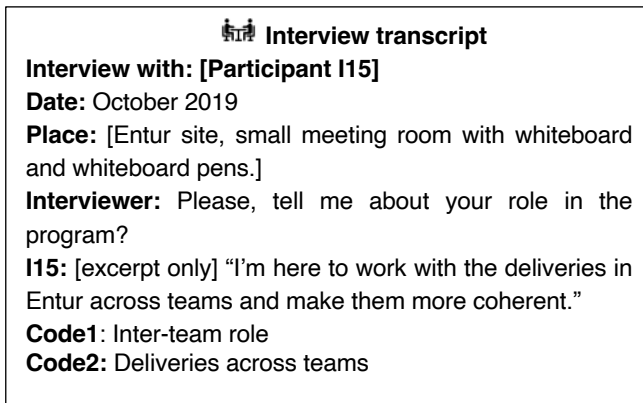


Fig. 3. Interview transcript extract

informally with the developers and other employees. In line with our ethnographic approach, we took detailed notes following all types of observation as well as after each day of fieldwork. Notetaking involved describing the physical setting, the artefacts used, and people involved, as opposed to focusing only on what appeared salient in any given situation [65]. We did this to capture the richness of the coordination activities conducted.

Figure 2 shows a sample note from a retrospective meeting.

Interviews. In addition to the extensive field observations, we conducted 31 semi-structured interviews. Twelve interviews were conducted in October 2018, two during April-July 2019, ten in October and November 2019, and seven in January 2020. On average, the interviews were 51 minutes long, on average. Informants held various roles relevant to inter-team coordination in Entur, such as team leaders, tech leads, and product owners, as well as the program architects and managers, and specialist roles such as the method and process specialist. Six of the participants were interviewed twice with one year in between.

Although the interviews were largely conversation driven, we used an interview guide to direct the conversation. The full interview guide is provided in Appendix A. Some standard questions asked were:

- *Can you tell me about your role on the project?*
- *What challenges do you see in this development program?*
- *How is information shared across teams?*
- *How is coordination conducted across teams?*

The interviews were recorded with the participants' consent and the first author transcribed them verbatim. Figure 3 provides a short excerpt from an interview transcript.

Supplemental material. As a final data source, we supplemented the observations and interviews with program documentation such as Slack logs, JIRA and Confluence documentation, and other resources such as meeting minutes and company presentations. Supplemental material was selected to reflect the period of the data collection. We had access to Slack, JIRA, and Confluence throughout the data

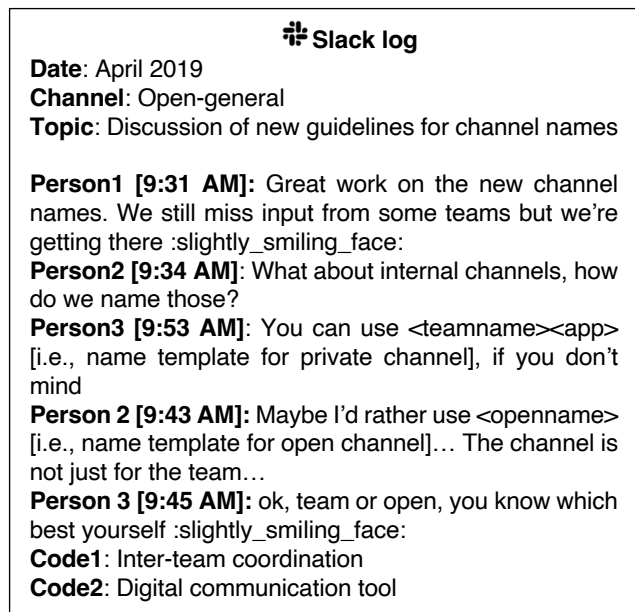


Fig. 4. Supplemental documentation extract: Slack log

collection period. For the purposes of the analyses, we only included material where aspects that are relevant inter-team coordination were discussed. As another example, we collected all available company presentations, as this material was less substantial than the chat logs and project documentation.

Examining these sources provided us with additional context related to, for instance, the use of coordination mechanisms, information about team organization, and inter-team documentation routines. For example, field notes from meeting observations were checked against meeting agendas when these were posted on Confluence, or Slack logs provided context to statements from interviews. Figure 4 provides a short extract from a Slack chat log.

3.3 Data Analysis

We analyzed the underlying data using thematic analysis [26], [27]. Thematic analysis is a method for systematically identifying and analyzing patterns across a data corpus, referring to all data collected for a project. Thematic analysis is suitable for handling large amounts of data, and therefore represented a suitable approach to handling the large data material resulting from the ethnographic approach to the data collection, including 113 hours of observation across 62 days of fieldwork, 31 interviews, and various forms of supplemental documentation.

Thematic analysis allows the researcher to identify commonalities across data items (e.g., an interview transcript or field note record) that are coded for meaning. The coded pieces of data are referred to as data *extracts*. These form the basis for the later identification of themes [26]. Figures 2-4 provide examples of extracts from each of the three data sources with codes.







Thematic analysis can be both inductively and deductively guided. When the analysis is inductively driven, themes have strong links to the data, whereas with the deductive approach, the existing literature guides the themes. Using a combination of both is common [27]. Both approaches guided our thematic analysis. During early analytical phases, we focused on the empirical data to derive the individual coordination mechanisms and group them into themes and patterns. During later phases, we focused on our understanding of coordination mechanisms from the existing literature, described in Sections 2 and 5. We used the qualitative data analysis software NVivo 12 for coding, and we kept a list of the coordination mechanisms identified in a spreadsheet that was later expanded to include the emerging framework. While sharing the full coding spreadsheet is not possible due to the underlying confidentiality clauses, we have shared several examples throughout the manuscript, summarized in Table 4.

3.3.1 Conducting the Thematic Analysis

Thematic analysis consists of six phases [26]: (1) familiarizing with the data, (2) generating initial codes, (3) searching for themes, (4) reviewing potential themes, (5) defining and naming themes, and (6) producing a report. Table 2 illustrates how we moved through the six analytical phases.

A theme “captures something important about the data in relation to the research question, and represents some level of *patterned* response or meaning within the data set” [26, p. 82]. A pattern relates to recurring instances of a similar type that are prevalent enough to be considered a theme. When a pattern or type is “enough” to constitute a theme is a judgment call on behalf of the researchers [26] based on questions such as, “What does this theme include and exclude?”

TABLE 2
PHASES OF THEMATIC ANALYSIS [23], [24]

Phases	How the phases were conducted
 1. Familiarizing with the data	We transcribed, read, and reread the material and noted down initial ideas on a regular basis throughout the data collection period. This familiarized us with the data to make initial analytical reflections on how inter-team coordination was performed. The 1 st , 3 rd and 4 th authors were involved in this phase.
 2. Generating initial codes	Initial codes were generated iteratively as data was collected. Figures 2-4 provide examples. During initial coding it is better to be too inclusive over too exclusive, as codes will be refined in later phases. From this, 59 potential <i>inter-team coordination mechanisms</i> were identified. The 1 st , 3 rd and 4 th authors were involved.
 3. Searching for themes	Codes were reviewed and refined to identify themes. The full data corpus was re-examined. Themes were related to the <i>categories</i> of inter-team coordination mechanisms as well as to the <i>underlying characteristics</i> of the coordination mechanisms. The 1 st and 2 nd authors were involved in this phase.
 4. Reviewing themes	Themes were checked in relation to the coded extracts and the entire data corpus. All identified inter-team coordination mechanisms were examined according to category and key characteristics. Their uniqueness was re-examined, and similar and overlapping mechanisms were identified. The number of mechanisms was reduced from 59 to 27. The 1 st and 2 nd authors were involved in this phase.
 5. Defining and naming themes	The specifics of each theme were refined and checked for coherence. Definitions and names were generated for each theme (see Table 3 and Figure 5). The 1 st and 2 nd authors were involved in this phase.
 6. Producing the report	Writing up the study provided a final opportunity to relate the analysis to the research questions and the literature, by the selection of compelling examples and illustrative quotes and iterating on the study presentation. All four authors were involved in this phase.

and “Does this theme tell us something useful about the data set and the research question?” [27]. In this study, we considered the categories and the characteristics of inter-team coordination mechanisms as themes.





Importantly, the thematic analysis process is iterative rather than linear, and moving back and forth through phases to ensure themes and patterns are related is encouraged [26], [27]. As such, elements of previous phases were involved in the later stages of the analysis. For instance, the full material was re-examined during Phases 4 and 5 to update themes and codes identified during previous phases.

3.3.2 Defining and Naming Themes

The first and second authors identified, reviewed, and defined the themes during Phases 3 to 5. One set of themes related to categories of inter-team coordination mechanisms. Through iterative discussions, the initial 59 coordination mechanisms were combined and reduced, resulting in 27 mechanisms. Among those, many shared similar features (i.e., they were of the same category). We therefore categorized the inter-team coordination mechanisms in three themes according to the category of the mechanism: meetings, roles, or tools and artefacts. More details on these categories are provided in Section 4.

A second set of themes related to the key characteristics of the coordination mechanisms. The socio-technical perspective on software engineering served well to capture the social and technical nature of inter-team coordination mechanisms.

TABLE 3
THE TOPS CHARACTERISTICS

 TECHNICAL	A characteristic of the coordination mechanism related to managing dependencies related to the software product itself. Also applies to digital tools or platforms supporting the development process. For example, the architect role or the tool Slack.
 ORGANIZATIONAL	A characteristic of the coordination mechanism that captures the wider structural context of the development organization, managing business process dependencies in particular. For instance, team design and organizational design.
 PHYSICAL	A spatial or tangible characteristic of the coordination mechanism. For instance, intangible mechanisms with spatial dependencies and physical artefacts and objects such as task boards.
 SOCIAL	An interpersonal or community-based characteristic of the coordination mechanism, related to the management of interpersonal dependencies. For example, roles or activities that enable coordination through groups, typically a meeting.

Through ongoing and iterative discussions, the first two authors examined each coordination mechanism in detail, discussing what made them social and technical based on how they worked to support inter-team coordination. All mechanisms were technical and social in nature. *Technical*, because all mechanisms related to either the software product or were technological tools or artefacts used to support software development, and *social*, because all mechanisms were interpersonal or community based. However, our case observations and analyses strongly indicated additional aspects that could not be explained using social and technical perspectives alone. From

these secondary analyses, two additional characteristics emerged; that is, the organizational and physical.

Some mechanisms displayed characteristics that captured the wider *organizational* context of the development process and activities. For example, the delivery process specialist role had as its primary goal to improve the inter-team delivery process, thereby managing process dependencies. Further, several mechanisms appeared to have spatial or tangible characteristics related to size-related or physical dependencies in the large-scale setting. For example, the platform and test teams would occasionally sit with the development teams to solve the relevant tasks. Further, most meetings ideally required appropriate meetings rooms. We therefore included a category to capture these *physical* characteristics. Definitions of the technical, organizational, physical, and social (TOPS) characteristics are presented in Table 3.

During the analyses, we also observed that most mechanisms could be placed under multiple TOPS characteristics. The first and second authors discussed all such occurrences to reach agreement on the mechanism's primary and secondary characteristics. Decisions were based on how the mechanism was used to manage inter-team dependencies and how it was represented in the data by the strongest evidence. For example, the weekly *Friday Demo* (see Table 4 and Figure 5) is an inter-team meeting that primarily serves a social purpose (demonstrated by an ☑ icon) and primarily manages knowledge dependencies (captured in the last 'description' column) in that teams take turns showcasing their work to all other teams. Another aspect reinforcing the social characteristic was the informal socializing following the demos. From seven demo observations, we could see how the demo often ended with casual

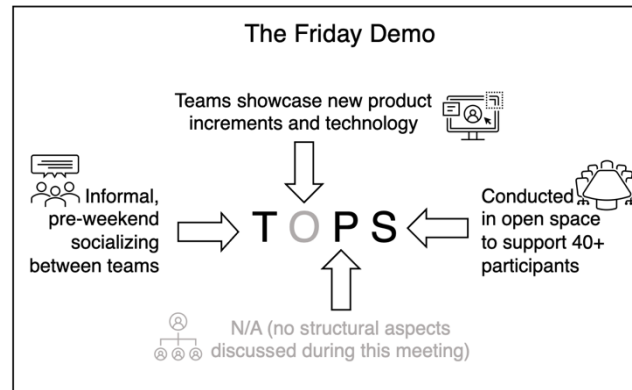


Fig. 5. The technical-organisational-physical-social (TOPS) framework, illustrated by the Friday Demo

conversations accompanied by some Friday snacks, providing people with an end-of-week break, and satisfying their informal socializing needs. Based on these observations, and because the demo's primary function was described in interviews as an informal arena for bringing people together before the weekend, we deemed the primary characteristic of the demos as being social. The demos also have technical characteristics in that they focused on the product and physical characteristics because they had to be conducted in the open office space to ensure room for all participants.

4 Case Study Results

In this case study, we set out to investigate how inter-team coordination mechanisms are used in large-scale agile software development and how these mechanisms support inter-team coordination. This section presents our findings.

From our analyses, we identified 27 inter-team coordination mechanisms across three categories: meetings, roles, and tools and artefacts. These form the taxonomy of inter-team coordination mechanisms, displayed in Figure 6. The three categories are further divided into six subcategories: (a) schedule meetings, (b) unscheduled meetings, (c) individuals playing specific roles, (d) teams playing specific roles, (e) tangible tools and artefacts, and (f) intangible tools and artefacts. The following sections are structured according to these categories. Table 4 provides brief details on all 27 mechanisms, their TOPS characteristics, and the ways they support inter-team coordination by relating them to the knowledge, process, and resource dependency categories [13] outlined in section 2.4.

4.1 Inter-team Coordination Meetings

Inter-team coordination meetings are meetings where team representatives and/or roles operating at the inter-team level discuss, coordinate, and share knowledge relevant across teams or to the development program as a whole. Inter-team meetings held at Entur included regularly scheduled meetings such as the team leader stand-ups, prioritization meetings, product owner meetings, program architect meetings, Friday demos, and the tech lead forum, which was the regular meeting of the tech lead community of practice. There were also retrospectives for team leaders, product owners, and tech leads, respectively (presented collectively in Table 4);

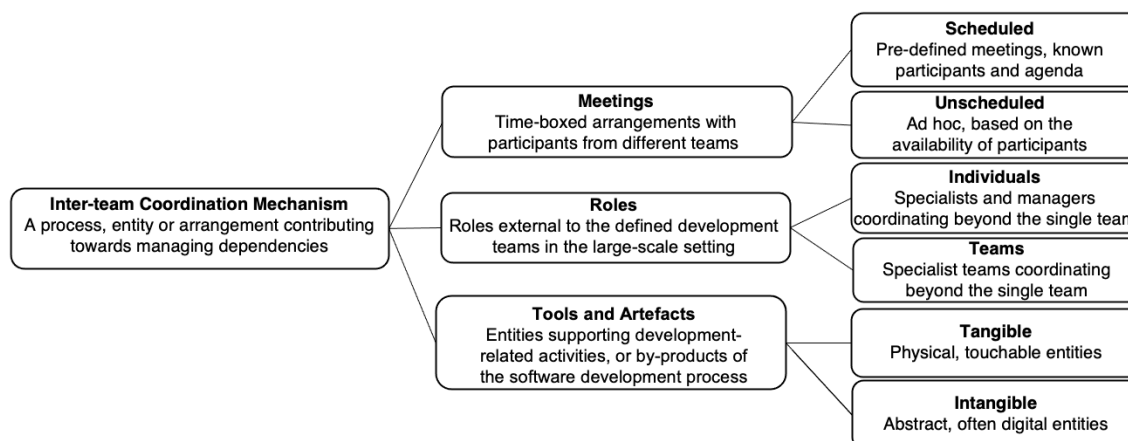


Fig. 6. A taxonomy of inter-team coordination mechanisms

quarterly product owner workshops; and Objectives and Key Results (OKR) workshops (to be explained in Section 4.3), where team representatives, program-level architects, and managers were present. In addition, there were various ad hoc coordination meetings. We therefore include both scheduled and unscheduled meetings in the taxonomy.

Table 4 describes the 10 inter-team meetings, their TOPS characteristics, and how they support inter-team coordination. All the meetings served to manage knowledge dependencies by enabling information sharing between teams, and fulfilling social needs, thereby displaying social characteristics. All meetings further served to address technical dependencies related to tasks or activities in terms of product features and/or requirements, development technologies, architecture, or similar. Some of the meetings also had an organizational purpose in that they served to manage dependencies related to the development process or business processes. Finally, all meetings had some physical requirements due to size and/or due to some physical artefact (e.g., a task board) that dictated where the meetings were held. To illustrate, we describe the team leader stand-ups and unscheduled meetings.

Team leader stand-up. Every week, the development, platform, and test team leaders gathered for a stand-up. The development manager, customer managers, and other managers also attended on an irregular basis to stay up to date, making for about 20 participants, on average, in each meeting. Facilitated by the agile method specialist, the meeting was focused on gaining an overview to identify current and upcoming dependencies that could cause blockages or delays across teams. The primary focus of these meetings was feature- or product-related progress across the teams; therefore, the meetings were characterized primarily as technical inter-team coordination mechanisms. One example was observed in March 2019, when a team leader raised the issue that the alerts that came into a dedicated Slack channel about technical issues in the production environment were “*not very clear.*” The team leader complained that many incoming alerts were difficult to understand, and accordingly were hard to prioritize. The question “*What are production errors, what are only alerts, and what can be ignored?*” was posed, followed by other team leaders joining in, starting a technical discussion about alerts’ definitions and framing.

The social characteristic can be illustrated by the community-based features, in that team leaders meet regularly to connect and update each other across teams, thereby managing

TABLE 4
INTER-TEAM COORDINATION MECHANISMS, PER TAXONOMY CATEGORY AND TOPS
CHARACTERISTIC

Coordination Mechanism	T	O	P	S	Description of the mechanism and how it supports inter-team coordination
COORDINATION MEETINGS (n= 10)					
Community of practice meetings*	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Team representatives meet bi-weekly to share topic-specific knowledge across teams, such as technical coordination, thus managing knowledge, process, and resource dependencies. Due to many participants, a large meeting room with many seats and audio-visual set-up is required.
Friday demos	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	A weekly demo for all employees. Teams take turn showcasing their work, demonstrating new features or ideas, thereby managing knowledge dependencies across all teams. An informal arena for socializing, often with snacks provided. Conducted in a large open space with audio-visual arrangements.
Inter-team retrospectives*	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Held approximately quarterly for discussing improvements of inter-team work processes, but also technical (product) or organizational aspects. As such, process as well as resource and knowledge dependencies are managed. Requires a room and tools suitable for retrospectives.
OKR workshops	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Held quarterly at an off-site location to discuss, align, set, and share inter- and intra-team OKRs, thereby managing knowledge dependencies. OKRs primarily relate to technical (product) progress, but can also be related to organizational outcomes, thus also managing process dependencies.
Prioritization meetings	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Bi-weekly, conducted in front of a prioritization task board. Focused on product and technical requirements, thus managing knowledge and resource dependencies.
PO weekly meetings	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	POs meet bi-weekly during lunch hours in a meeting room close to the cantina. Discussion of technical product, as well as organizational topics, managing resource, process, and knowledge dependencies.
PO workshops	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	POs meet quarterly to plan and discuss longer-term technical product-related areas. Organizational issues, such as team structure, are also discussed. Held at an off-site location and includes retrospectives and informal socializing. The workshop thus manages resource, process, and knowledge dependencies.
Program architect meeting	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Weekly meeting where product technical and architectural quality are recurring themes. Organizational aspects can also be discussed, thereby managing primarily resource, but also process dependencies.
Team-leader stand-ups	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Weekly stand-up for sharing status across teams, with a focus on product, thus managing knowledge and resource dependencies. Conducted in open space.
Unscheduled meetings	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Conducted ad hoc, as needed, across relevant teams. Typically focus on product feature and deliveries, thus managing knowledge and resource dependencies. Held in open office space or meeting rooms.
COORDINATION ROLES (n = 9)					
Customer managers	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	One per major customer, this role attends meetings at the clients' sites. Brings information on e.g., requirements and specification back to the teams, thus managing knowledge and resource dependencies.
Development manager	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	Responsible for team leaders, has a high-level overview of teams' major tasks and prioritizations. Also responsible for staffing, thus involved in managing resource (entity) and business process dependencies.
Agile method specialist	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	Responsible for agile methods and has overview of requirements, tasks, and prioritizations across teams, thus managing process and technical resource dependencies.
Platform team	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Internal service team that manages technical resource dependencies by facilitating the teams' technical environment, providing a common platform. Some facilitation requires sitting with development teams.
Delivery process specialist	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	Implements an inter-team delivery process with the goal of aligning and improving inter-team product deliveries, thus managing primarily process but also resource, dependencies.
Program architects	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	Concerned with the inter-team software, product, and organizational architecture. Involved in technical and structural discussions, thus managing resource, business process, and knowledge dependencies.
Product manager	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	Responsible for POs, has overview of requirements and prioritizations across teams and clients. Involved in structural discussions, thus managing entity resource and business process dependencies.
Task force teams	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Temporary teams consisting of members from permanent teams used to implement interdependent features of high priority, thus primarily managing resource (technical and entity) dependencies. The team is co-located while working together, and dissolves after feature completion.
Test team	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Performs testing across teams and coordinate inter-team testing efforts, thus managing process and resource dependencies. Some testing requires sitting with the development teams.

COORDINATION TOOLS AND ARTEFACTS (n = 8)			
Communication tools*	<input checked="" type="checkbox"/>	✓	Tools such as e-mail and Slack, enabling digital communication and information sharing across teams, thus managing resource (technical) and knowledge dependencies.
Documentation tools*	<input checked="" type="checkbox"/>	✓	Tools such as JIRA and Confluence, supporting the development process and enabling information sharing across teams, thus managing resource (technical) and knowledge dependencies.
OKRs ^a	<input checked="" type="checkbox"/>	✓ ✓ ✓	Conveys information on both technical (product) and organizational objectives and outcomes across teams, thus managing primarily resource, but also business process and knowledge dependencies.
Burndown chart ^a	<input checked="" type="checkbox"/>	✓	Digital, displays information related to completion of product-related development tasks and activities across teams, thereby managing resource (technical) and knowledge dependencies.
Prioritization document ^a	<input checked="" type="checkbox"/>	✓	Digital, displays information on overall development priorities, across teams and clients. Enables communication and information sharing, thus managing knowledge, but also resource dependencies.
Roadmap (digital) ^a	<input checked="" type="checkbox"/>	✓	Enables communication and information sharing related to overall product delivery milestones across teams, thus managing resource, process, and knowledge dependencies.
Roadmap (physical) ^a	<input checked="" type="checkbox"/>	✓ ✓ ✓	Similar as the above, but displayed in the open office space, thus containing less detail than the digital roadmap. People engage with it physically, e.g., by updating tasks.
Task board ^a	<input checked="" type="checkbox"/>	✓ ✓ ✓	Similar characteristics as the prioritization document but displayed in the open office space therefore showing top prioritizations only. People engage with it physically, e.g., by updating tasks.

Notes. An asterisk (*) indicates that similar mechanisms were collapsed into one. PO = Product Owner. Primary characteristic, indicated by , is set based on which type of dependency is primarily managed. Artefacts are indicated by ^a. The types of dependencies, following (Strode, 2016), are described in section 2.3.

knowledge dependencies and serving a social purpose by connecting team leaders. Because the scope of the stand-up was brief and focused, topics of structural or wider organizational nature typically were not discussed. Finally, there were also physical requirements connected to the meeting, in that the number of participants created a requirement for enough open office space for about 20 people to stand in a circle and at the same time not interfere with the developers' work.

Unscheduled meetings and ad hoc coordination. In addition to the many scheduled meetings, unscheduled meetings were used extensively to resolve day-to-day inter-team dependencies. *“Oftentimes, we solve things by walking over and talking to each other. I really like that. Not everything needs to be a meeting”* [I01, Product Owner]. At other times, it was necessary to assemble more people.

During our 62 days on-site, we observed many instances of unscheduled meetings, and we were invited to join several of these. As an example, on one occasion in April 2019, we witnessed over the course of a day how one team needed to coordinate with three other teams they depended on for completing a feature (i.e., resource dependencies). Early in the day, the team leader talked to his team to gain an overview of issues that needed to be resolved and to identify any dependencies on other teams that could delay the work. Following this, the team leader disappeared for a while, to come back having gathered representatives for the relevant teams for a meeting to address the technical dependency across the three teams that had blocked the developers' progress.

The technical characteristics of unscheduled meetings are evident in that their primary purpose was to manage technical dependencies by quick product- and task-related coordination. These meetings further bear a strong social characteristic due to the interpersonal nature of such meetings. Physical aspects were also evident. While coordination may be performed digitally, having people nearby was considered valuable for swift dependency management. *“You achieve*



Fig. 7. A multi-purpose room at Entur, used for ad hoc coordination, socializing, and for meetings including a physical task board.

much more by just talking to people face-to-face than spending time writing on Slack or sending e-mails” [I08, Product Owner]. Moreover, ad hoc physical coordination required suitable spaces (see Figure 7).

4.2 Inter-team Coordination Roles

Inter-team coordination roles were regarded as roles external to the development teams. Table 4 presents these nine roles and their TOPS characteristics. We include both individual roles (i.e., the expert and manager roles) and team roles in our taxonomy. All roles are performed by people coordinating with other people at an inter-team level, thereby serving to manage knowledge dependencies and holding social characteristics. However, they also had different purposes. In the following, we describe the nine roles and their characteristics in more detail.

The expert roles. While all inter-team roles contributed to managing knowledge dependencies, expert roles were primarily important for managing technical dependencies. These roles included the program architects, the agile method specialist, and the delivery process specialist.

The *program architects* were senior architects who held detailed knowledge about Entur’s technical and organizational architecture. As such, they were important for managing knowledge expertise dependencies, for instance, by sharing information across teams regularly during the tech lead forum (one of the communities of practice) meeting and in the tech lead Slack channel and Confluence page. While the program architect role was primarily technical, they also had a wider organizational purpose: “*My role includes having an overview of questions like, ‘How are we organized?’ ‘What do we measure?’ ‘Are we data-driven in our work?’ And one of my earliest initiatives when starting here was establishing an architecture group to achieve more than each individual [architect] can do alone*” [I19, Program Architect].

The *method specialist*, responsible for agile methods and practices, was important for managing dependencies across the program. The role was described as “*a jack of all trades, really, who see needs and I try to fill them*” [I20, Manager]. For instance, the method specialist implemented artefacts (e.g., the inter-team backlog), facilitated inter-team meetings (e.g., stand-ups and retrospectives), and introduced the Friday demos exemplified in Section 3.2.2. Finally,

primary goal of the *delivery process specialist* was improving the inter-team delivery process, thereby attaining organizational needs and managing process dependencies by “*making the deliveries more aligned and contribute to improved predictability*” [I15, Manager].

The manager roles. The product manager, development manager, and customer managers were important for managing entity and business process dependencies, and primarily held organizational characteristics.

The *development* and *product managers* each had personnel responsibility for the team leaders and product owners, respectively, and were responsible for coordinating these groups. As such, they both managed resource dependencies. “*They work to get more resources, recruit their own people [i.e., team leaders], and make sure they are developed*” [I12, Manager]. Perhaps more importantly, they were part of organizational discussions and decision-making, making them important in relation to business process dependency management “*to look at processes and routines so that everyone can work effectively. The goal is to make the hottest development environment in the country!*” [I12, Manager].

The *customer managers* were considered primarily important in relation to technical dependencies, but also knowledge dependencies, in that they collected and shared technical information between the customers and the teams. “*In practice, they are part of defining what we promise the customers*” [I20, Manager]. The physical characteristic also applied to the customer managers because they were required to spend time at the clients’ offices.

The team roles. Both internal support teams such as the *platform* and *test teams* and the *temporary task force teams* were important for managing technical dependencies. These teams specifically targeted development activities across teams and contributed to coordinating product-related issues above and beyond the single teams.

The test team “*coordinate[d] test runs and supports the teams with test automation*” [I09, Product Owner], while the platform team provided various shared services and infrastructure across the development teams. As Entur continued to scale, the platform team was central to technical dependency management as “*almost everything runs through the platform team*” [I22, Tech Lead]. The team leader of the platform team explained: “*We’re a bit all over the place because of our position. We work across teams, and we’re a technically heavy team, which means that we notice a few things that need to be coordinated across teams*” [I25, Team Leader]. In addition to managing technical inter-team dependencies, there were physical characteristics related to both teams, as team representatives would often sit with the relevant development teams they were supporting.

As a third type of team role, Entur used temporary teams as needed. These were known as “task force teams” and consisted of members from different development teams who were assembled to implement inter-dependent product features of high priority. As such, the technical characteristics are illustrated by the teams’ focus on addressing product-related needs and requirements. The task force teams were co-located and held their own agile routines. “*Once we have established shared priorities, there’s a pretty good flow. We set up stand-ups and arenas to coordinate, and there is a lot of communication*” [I22, Tech Lead]. When their

tasks were completed, the task force teams dissolved, and the members returned to their original teams.

4.3 Inter-team Coordination Tools and Artefacts

We consider inter-team coordination tools and artefacts as objects that serve to manage dependencies between development activities across teams. At the inter-team level these are broad, and a bit distant from the primary development activity of writing code (as this primarily happens at the team level). We identified two types of tools and six artefacts specifically used for coordination across teams in Entur (see Table 4). In the taxonomy, we categorize these as *tangible*, material entities, and *intangible*, digital entities.

In software engineering, a tool, broadly speaking, is used to support development-related activities. Two types were used: communication tools, such as Microsoft Teams, Google Workspace, and Slack, and documentation tools such as JIRA and Confluence. An artefact is typically considered a tangible by-product of the software development process, such as a task board on a wall (see Figure 7). Artefacts can also be digitally represented, as is often the case with program documentation. We included the inter-team task board, physical and digital roadmaps, prioritization document, burndown chart, and OKRs as artefacts.

All identified tools and artefacts supported coordination across teams by managing technical resource dependencies as their use was connected to developing the technical product (six of these mechanisms were also technologies in themselves). They also served to manage knowledge dependencies in light of their social characteristics as collaborative tools. Additionally, some tools and artefacts were physical entities, such as the various task boards that people engaged with as well as a physical roadmap that was displayed in the open office space. A few of these tools and artefacts were used to manage business process dependencies, however, OKRs held such organizational characteristics as they were related to process dependencies as well as technical dependencies. We now present two illustrative examples, Slack, and OKRs.

Communication Tools: Slack. While there were several options for digital communication available at Entur, Slack was allegedly by far the most used communication platform. Slack is a digital collaboration tool that allows users to communicate in public or private group channels as well as with private direct messages [20]. Slack’s overall purpose at Entur was enabling swift and timely digital communication among individuals and teams working together in the development program, thereby contributing to managing knowledge dependencies. During an interview, a team leader who had been with the program since the outset explained that they had “*always used Slack,*” at first mostly within the teams, but that “*now you have a lot of channels across teams. All teams have their own open channel that others external to the team can use, and there is a lot of activity in those channels*” [I13, Team Leader].

Primarily used for written communication, Slack also allows for video chats, file sharing, and the set-up of bots known as Slackbots that perform various tasks, such as giving production error alerts, but also “*bots to notify people ‘now it’s stand-up!’ or ‘now’s demo time!’*” [I13, Team Leader]. Slack was primarily used to resolve technical dependencies by means of written communication. In addition to the open team-channels, there were specific channels set up for

inter-team coordination. For instance, the tech leads had their own channel, as did the team leaders and product owners. In addition, there were several topic-specific inter-team channels, such as the open discussion channels “techtalk” and “ux-design” that effectively provided a means for coordinating across teams.

In the TOPS framework, Slack is primarily characterized as technical, as most communication (be it human or bot-driven) is focused on product development. However, Slack also served to fulfill social needs by connecting people, particularly if someone was working off-site.

Objectives and Key Results. In short, OKR refers to a goal-setting process framework focusing on creating attainable goals and outcomes, emphasizing employee involvement and bottom-up participation [66]. The result of this process was that specific OKRs that summarize the objectives (i.e., a description of some qualitative goal) and key results (i.e., quantitative goal statements) set for a certain period [67]. In our findings, we consider the OKR framework as a coordination tool, and the specific OKRs, that is, the output of the framework, as coordination artefacts.

Entur started using the OKR framework in 2019. *“We needed a fresh start. To do something differently, structurally, than the former goal metric. What I like about OKR is that it breaks goals down from strategies to tasks”* [I03, Manager]. They implemented the framework iteratively, starting with the product owners and managers in a pilot run during spring 2019, and included the team leaders from fall 2019. Using OKRs served to coordinate goals across teams. *“You can see it through the synergies resulting from sharing objectives and key results between teams”* [I03, Manager].

The OKRs’ primarily related to managing technical dependencies. *“In the architect group, we have an OKR that is ‘to make the technical state across teams known’. This represents a way to capture technical issues and respond to them”* [I14, Program architect]. The OKR framework also served to manage knowledge dependencies and has social characteristics because representatives from the different teams work with developing OKRs collectively. Further, some OKRs, such as the managers’, were directed at organizational purposes by managing business process dependencies.

While the OKRs in themselves are intangible artefacts, there were physical requirements related to their formation and use. The OKR workshops needed to be held off-site, as there was not enough office space available to host all participants (more than 30 in each workshop). Furthermore, to effectively serve to manage knowledge dependencies, and to be followed up on, the OKRs should be visible. This was also related to physical aspects in that *“we must acknowledge that we do not give them enough day-to-day focus [...] I think we need to place them on a wall to be reminded that ‘This is what I’m supposed to work on’”* [I06, Manager].

5 Discussion

Coordination and coordination mechanisms have been subject to much research scrutiny within software engineering in both distributed and co-located settings. Previous research has shown that dependency awareness is crucial to the success of inter-team coordination in large-scale

agile, by allowing teams to plan and align their development activities [6] and to handle the many coordination challenges in large-scale agile [1], [9]. In this study, we have continued this line of research by investigating the research question “*Which inter-team coordination mechanisms are used in large-scale agile software development, and how do these mechanisms support inter-team coordination?*” This investigation resulted in a description of 27 inter-team coordination mechanisms (Table 4), that were used to develop a taxonomy of inter-team coordination mechanisms (Figure 6) and a framework for describing the characteristics of these mechanisms (Table 3, Figures 5 and 8).

Our research was motivated by the notion that although previous research has focused on describing the coordination process and through this has identified and described coordination mechanisms in use, there exists no comprehensive collection of inter-team coordination mechanisms to guide research and practice. Additionally, our understanding of the underlying characteristics of such mechanisms that dictate their practical implementation remains limited. With this study, we contribute to filling these gaps.

5.1 A Taxonomy of Inter-Team Coordination Mechanisms

As the first contribution, we propose a taxonomy of inter-team coordination mechanisms. By this, we provide a tool for identifying and evaluating mechanisms to guide research and practice on inter-team coordination. The taxonomy includes three main categories that includes a total of six sub-categories: *scheduled* and *un-scheduled* meetings; *individual* and *team* roles; and *tangible* and *intangible* tools and artefacts (see Figure 6).

Taxonomies provide value by their ability for sensemaking in relation to the meta-category, the extent to which inferences can be made from it, and the extent to which it is useful within its domain [56]. The taxonomy of inter-team coordination mechanisms contributes to earlier taxonomies on dependencies and coordination mechanisms [8], [13] by extending the focus to the inter-team level, and to the knowledge domain of large-scale agile. However, empirically derived taxonomies should be evaluated against existing quality criteria, and compared against existing literature [56]. Therefore, we will first assess our taxonomy against existing evaluation criteria before we relate our findings to the existing research on inter-team coordination mechanisms outlined in Section 2.

5.1.1 Assessing the taxonomy against existing criteria

As an overall criterion, taxonomies should be organized around a single meta-category [57]. Our proposed taxonomy meets this criterion with its focus on inter-team coordination mechanisms. Taxonomies should further be evaluated against predetermined quality criteria to assess their appropriateness and relevance [56], [57]. In the following, we evaluate the taxonomy against Nickerson et al.’s criteria of conciseness, robustness, comprehensiveness, extendibility, explanatory ability and usability [56].

Because the taxonomy contains a limited number of dimensions, i.e., the four categories with a total of six sub-categories, it meets the criterion of *conciseness*. The included categories appear sufficient to capture all inter-team coordination mechanisms observed from our data. The categories are further mutually exclusive, i.e., a meeting is sufficiently different from a tool. Thus, the *robustness* criterion is met. While our categories can contain all objects in the

empirical case, it is possible that future research will discover additional categories. More research using the taxonomy is needed to meet the *comprehensiveness* criterion. Related to the above point, the *extensibility* criterion holds that the taxonomy must allow for the extension and addition of new categories as research progresses. Should new categories be needed based on new empirical observations or studies, there is room to add these as applicable. The taxonomy is thus extendible. Our categories *show explanatory ability* as they are intuitive enough so that others may readily use them to classify coordination mechanisms observed in other cases. However, this criterion will be fully met once other studies have been conducted using the taxonomy. The final criterion, *usability*, is met if, over time, the taxonomy is used by others within the domain. As such, while we hope the taxonomy proves usable, future research on inter-team coordination will demonstrate whether this criterion is met over time.

5.1.2 Relating the taxonomy to existing studies

To illustrate how the taxonomy can be used to with existing research, we relate the taxonomy categories to a selection of studies of inter-team coordination in large-scale agile, summarized in Table 5. For the purposes of this illustration, we narrowed our focus to studies published in peer-reviewed journals no earlier than 2015. As such, this list is non-exhaustive.

Meetings. In our findings, both scheduled and unscheduled meetings contributed to managing inter-team dependencies. Inter-team meetings supported inter-team coordination by managing knowledge dependencies and process dependencies, as the meetings contributed to sharing information and knowledge about the product and the development process across teams. Meetings further provided inter-team representatives with access to the information held by expert roles, thus managing resource dependencies related to the availability of these roles.

Table 5 shows previous research that has focused on meetings in inter-team coordination. For example, a study by Dingsøy and colleagues on coordination in multi-team development programs [48] found that meetings such as demos, retrospectives and board discussions (similar to the task board meetings in our results) contributed to managing dependencies in the large-scale program by enabling knowledge sharing and promoting overview across teams, for instance by avoiding teams working on the same part of the codebase [48]. In their research on

TABLE 5
SELECTION OF STUDIES ON INTER-TEAM COORDINATION IN LARGE-SCALE AGILE
(M= Meetings; R=Roles; T&A=Tools and Artefacts)

	M	R	T&A
Bass, 2015		✓	
Bass & Haxby, 2019		✓	
Dingsøy et al., 2017	✓		✓
Dingsøy et al., 2018	✓		✓
Kettunen & Laanti, 2017		✓	
Moe et al., 2018	✓		
Paasivaara et al., 2018	✓		
Paasivaara & Lassenius, 2019	✓		
Sablis et al., 2020	✓	✓	✓
Shastri et al., 2021		✓	
Smite et al., 2019	✓	✓	✓
Stray & Moe, 2020	✓		✓
Our study	✓	✓	✓

large-scale agile frameworks in Ericsson, Paasivaara and colleagues [3], [21], and Smite and colleagues [23], show how communities of practice can be used to support inter-team coordination across a wide range of purposes. In line with how Entur used their communities of practice, Ericsson used such to learn and share knowledge between inter-team roles [3], to coordinate technical work and for developing the organization [21].

Unscheduled meetings have also been demonstrated to facilitate inter-team coordination. A second study by Dingsøyr et al., [5] showed how a large-scale program increasingly used such informal coordination arenas to resolve emerging coordination needs. Similar results were found in another study on scheduled and unscheduled meetings, where the category of meeting used depended on the maturity of the development organization and the experience of the participants [15].

Coordinator roles. Our results show that both individual and team roles perform important functions for inter-team coordination. Both expert and manager roles contribute to managing entity resource dependencies, as their overview of technical and business process dependencies make them important inter-team roles, and developers' access to these roles are important for resolving dependencies across teams. Table 5 displays previous research that has focused on roles in inter-team coordination.

Manager roles are characteristic at the inter-team level in large-scale agile. In our data, the product manager, development manager and customer managers were important for managing dependencies at the inter-team level. Large-scale agile projects differ, and which roles operate at the inter-team level may vary. For example, Shastri et al., [53] describe the project manager role in coordinating between agile teams [53]. Bass focused on the functions [54] and activities [68] performed by product owners, showing how this role is an important role for inter-team coordination. In our case organization, product owners were considered part of the development teams and were therefore not included as inter-team coordination mechanisms. However, in other organizations, the product owner role may be external to the teams [54]. The taxonomy is flexible enough to handle such context-specific aspects.

Sablis et al. [47] and Kettunen and Laanti [51] both point to the importance of expert roles for team-external coordination. In their studies, the architect role was highlighted as particularly important for managing dependencies related to technical coordination across teams. This is in line with our results, where the expert and manager roles were found closely linked to dependency management, in particular related to the availability of their knowledge.

In addition to conceptualizing roles as an inter-team mechanism, our taxonomy contributes with the category of team roles such as platform and test teams. Team roles are not included as coordination mechanisms in any of the selected studies. Our results show that such teams are important for example in managing dependencies in assuring technical alignment across teams, and that they should therefore be included as coordination mechanisms. Future research should aim at uncovering more knowledge about these types of teams.

Tools and artefacts. In addition to meetings and roles, we found that both *tangible* and *intangible* tools and artefacts were important for managing dependencies related to the development process. Knowledge dependencies between teams were managed for instance by shared task boards and roach maps enabling overview. The use of OKRs, as well as shared

collaboration and documentation tools, such as Slack and Confluence, contributed to alignment across teams, thereby managing technical dependencies arising from the development process. The final column in Table 5 shows that tools and artefacts have also been in focus in existing research.

Dingsøy and colleagues [5], [48] report that the use of instant messaging, masterplans, guidelines and wikis were important impersonal coordination mechanisms. These compared to the roadmaps, prioritization documents and documentation tools used in our case. In line with their findings, we found that these tools and artefacts needed to be flexible and adaptable to reflect the fast-paced development process [48]. Further, the use of instant messaging tools were used for knowledge sharing across teams, in particular related to technical issues, but also process-related and even informal communication [5].

Among existing communication tools, Slack has received recent research attention as a useful coordination mechanism. A recent study shows that Slack enables frequent and timely knowledge sharing, but that its efficiency as a coordination mechanism depended on a shared understanding about how to use the tool across teams [20]. Our results show that the use of Slack's features, such as dedicated channels and Slackbots successfully supported knowledge sharing and communication across teams, indicating that such practices may be success factors for digital coordination.

None of the selected studies include the use of OKRs. This may be because OKR is a relatively new framework. In our findings, OKRs represent both tools and artefacts, in that the OKR framework provides a coordination tool for efficiently managing dependencies across teams both by the OKR framework itself and by the specific artefacts (i.e., the specific OKRs) resulting from using the tool. As the popularity of the framework is growing [67], future studies should further explore OKRs in relation to coordination in agile organizations.

5.2 Extending the Socio-Technical Perspective

A second contribution of our study is a framework describing key characteristics of inter-team coordination mechanisms. The TOPS framework, presented in Table 3 and Figures 5 and 8, is inspired from ideas of software engineering as a socio-technical practice which has a long historical context [69]–[71]. We believe the TOPS framework can be used as a guiding lens for research to analyze the coordination mechanisms used in any large-scale setting (i.e., co-located, distributed or a hybrid) to better understand the coordination practices used in the specific organization.

5.2.1 The TOPS characteristics

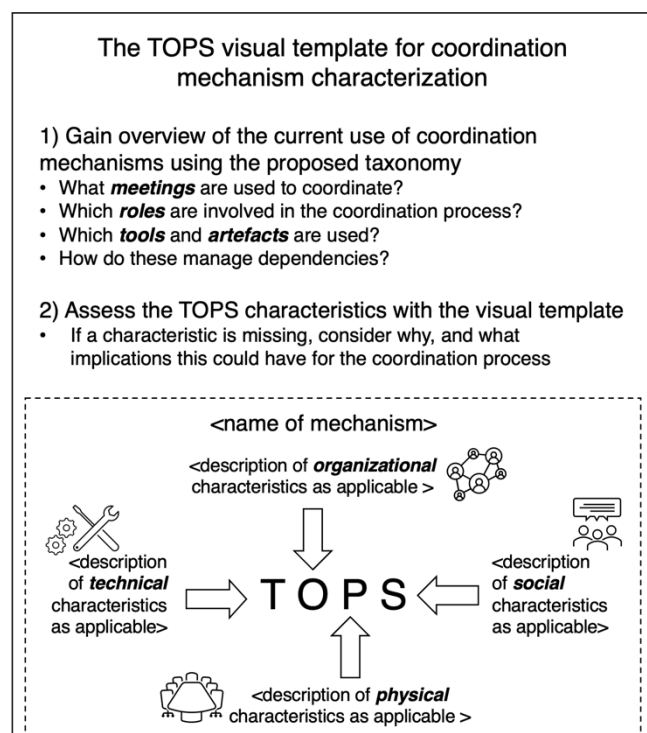
This study was conducted with the awareness that coordination in software development is performed using mechanisms that are socio-technical in nature. Indeed, in most contemporary organizations, the interactions between human, and thus social, and technological aspects are interlinked to such an extent that it is increasingly difficult to study one aspect without the other [71], [72]. The socio-technical perspective offered a lens that enabled studying software development including both the technical details of the tasks and technologies and the social and human characteristics of the people involved.

A key finding is that these mechanisms were not limited to “social” and “technical” aspects. Our results indicate that these two characteristics may be too narrow to capture the complexity and level of detail of modern organizations [73]–[76], in particular in large-scale agile software development. This is demonstrated by the 27 inter-team mechanisms displaying at least two, often more characteristics, including organizational and physical.

Some authors have suggested a socio-technical matrix dividing the social subsystem into “people” and “structure,” and the technical subsystem into “tasks” and “technology” [74]. Others have suggested including cultural, organizational, and collaborative perspectives to the socio-technical analysis [76]. In a related vein, our findings suggest that to understand coordination in large-scale software development, there may also be a need to understand the complex interplay between technical, organizational, physical, and social aspects of coordination. Based on our analysis, the majority (i.e., 21 out of 27) of mechanisms primarily held technical characteristics. This is not surprising, given that the case’s overall purpose was developing software. As such, the purpose of most coordination mechanisms was to manage technical dependencies, requirements and needs across teams. The social characteristics were most evident in mechanisms that managed knowledge dependencies. Additionally, mechanisms such as meetings and collaborative tools also served to fulfill peoples’ social needs, thus reinforcing the social characteristic. In addition to the social and technical, another two characteristics, organizational and physical, could be associated with the inter-team coordination mechanisms.

The organizational characteristic relates to properties (i.e., requirements or purposes) of a mechanism that captures the development activity’s wider structural context. In our results, this characteristic was often associated with mechanisms that managed process dependencies. This applied in particular to manager roles directly involved in structural discussions. Their primary purpose was to provide a formal structure and make decisions on team organization, including deciding whether new teams should be formed or assessing the existing team set-up. However, most mechanisms held it as a secondary characteristic. For example, we observed inter-team retrospectives where organizational issues such as how to arrange the teams for optimal delivery or collaboration with the other business areas in the organization were discussed and resolved.

The high number of people involved in our large-scale case organization required managing size-related dependencies. As such, several mechanisms appeared to have *physical*, that is, spatial or tangible characteristics. In our results, this often related to meetings in the form of spatial requirements to fit all participants, or to the platform and test teams physically sitting



with the teams they were supporting. In line with previous research on coordination in large-scale agile, open spaces and meeting rooms appeared key to enabling ad hoc coordination [5], [48]. The physical characteristic also related to the tangible nature of artefacts, such as inter-team task boards and visual representations of roadmaps and prioritization lists, important for managing knowledge dependencies across teams.

Large-scale software development encompasses not only human and technical aspects, but also aspects of the surrounding organization in which software development takes place [55]. As such, social, technical, organizational, and physical characteristics may intertwine. Further, coordination mechanisms are not static, stable entities. Rather, they are formed and re-shaped as they are used to fit the given coordination needs present in a given situation [35], as research on inter-team coordination in large-scale agile indicates [14], [48]. Accordingly, the TOPS characteristics are also conceivable to change and evolve over time. This constitutes an interesting avenue for future research.

5.2.2 The TOPS framework and “Work from anywhere.”

We concluded our data collection in January 2020. The TOPS characteristics are based on how the mechanisms appeared at the time in the co-located development program. Shortly thereafter, the global outbreak of COVID-19 forced organizations worldwide to go digital “overnight.” As a consequence, the digital office first replaced then later complemented the physical [77]–[79].

We believe the current “work from anywhere” (WFX) situation presents an opportunity to illustrate how the TOPS characteristics reflect the changing and dynamic nature of coordination mechanisms and their underlying characteristics. For example, WFX resembles the setup of distributed teams. Research conducted prior to the pandemic indicates that coordination is more challenging in distributed compared to co-located settings [80]–[82]. The TOPS characteristics – especially the physical – can enable focused research investigations into how inter-team coordination may change in WFX contexts. With respect to technical coordination, prior work suggests that working digitally does not have significant detrimental effects on coordination effectiveness. Most tools and artefacts can be digitally represented, and developers are accustomed to coordinating with digital tools [20]. While research conducted early during the pandemic showed negative effects between well-being and productivity [77], software engineers are still able to perform their work and coordinate with others [79]. Indeed, relating to the technical characteristics, a recent study showed that the interest in and use of pair programming practices increased during the first year of working from home, due to the practice’s technical and social characteristics [78].

We believe the physical characteristic can be particularly important in the current work environment. Relating to the social and organizational characteristics, most meetings can be held via virtual means. Further, mechanisms with clear organizational and physical features, such as OKR workshops requiring large meeting spaces, can be conducted digitally. While social contact and work coordination can be carried out via digital tools such as Microsoft Teams and Zoom, spending hours in digital meetings leads to increased fatigue [83], and a lack of social contact with colleagues can lead to negative psychological and well-being issues [77], [79]. As such, the physical characteristics are most prominently felt in their absence.

5.3 Implications for Practice

As a third contribution, the proposed taxonomy of inter-team coordination mechanisms and the TOPS framework provide a knowledge base and a structured approach for software practitioners to understand and improve inter-team coordination mechanisms in large-scale agile.

The taxonomy and the TOPS framework are sensitive to context, as the list of coordination mechanisms included in our analyses (i.e., the 27 mechanisms Entur used) are not the only possible mechanisms for large-scale agile coordination. The taxonomy's three categories and six sub-categories provide a robust structure to understand and further investigate coordination mechanisms used in most software development organizations. For instance, organizations following SAFe will use specific inter-team coordination mechanisms such as the *agile release train* (a process tool) and *the release train engineer* (a role) [51]. Other large-scale programs following a more hybrid approach may not have the same labels on their coordination mechanisms, but still have the same functions performed. The taxonomy and the TOPS framework may be particularly relevant in the current global WFX situation where many inter-personal meeting arenas have been replaced with virtual spaces. Our taxonomy can thus be extended to include more relevant inter-team coordination mechanisms.

Practitioners can use the taxonomy in Figure 6 to identify which mechanisms are used for inter-team coordination. From this, it is possible to identify the applicable TOPS characteristics following the definitions in Table 3. For example, inter-team representatives could map areas where there are coordination needs, what coordination meetings are used, which roles are involved, and which tools and artefacts are being used. This could be categorized in the taxonomy of inter-team mechanisms, providing the organization with a structured overview of their coordination situation. From this, representatives could further assess the underlying characteristics with the TOPS framework, as illustrated in Figures 5 and 8. Such an assessment could result in a detailed picture of the mix and balance coordination mechanisms and characteristics, and an overview of the organization's current coordination strategies. The organization could further evaluate whether this picture appears well-suited for addressing the organization's coordination needs. For instance, having a large portion of mechanisms requiring physical coordination would perhaps not be optimal in a distributed environment. Figure 8 provides a visual template that can be used as a tool to support this process.

We believe that the TOPS framework can offer practitioners a useful thinking and visualization tool for assessing and improving coordination practices. Our study indicates that visualizing which mechanisms are in use, as well as their defining characteristics, can help to provide an overview of the coordination setting and which mechanisms are being used for each purpose. We further believe the visual template can serve to illustrate situations where dependency management is lacking by the absence of mechanisms with the desired characteristics.

6 Limitations and Evaluation

Our study is a qualitative, single-case study, and we therefore consider the study's limitations in relation to criteria applicable to such studies [62], [84], [85]. Case studies can adopt different

philosophies, including an interpretivist stance [61], [62], as in this study. Interpretive case studies provide rich opportunities for describing real-life phenomena [62], and serve well as the basis for taxonomy building, because of the closeness to the data required of such studies [56]. To ensure a systematic and rigorous research process, we employed a range of quality assurance procedures. In the following, we review some of the study's potential limitations in relation to the quality criteria of credibility, transferability, and confirmability [84] that is much used in interpretive and constructivist qualitative research, including in the software engineering field, e.g., [63], [86].

Credibility. The ethnographic approach to collecting the data served well to generate a rich and diverse data material, carefully collected over a relatively long period of time [64], [65]. In collecting our data, we relied on observational protocols and semi-structured interviews. The reliance on several data sources (i.e., data triangulation) further strengthen the credibility of our analyses [61], [85]. While the application of the ethnographic approach was limited to the data collection procedures, the thematic analytical process ensured a rigorous, yet flexible analysis to generate the findings. To ensure rich data collection and triangulation of interpretations, the first, third, and fourth authors regularly discussed insights gained during field work and involved the second author in the data analysis and the taxonomy and framework development. Finally, member checks with Entur representatives provide additional trustworthiness to our findings [65], [85].

Confirmability. The primary advantage of interpretive case studies is that they encourage deep immersion in the data. While this may protect researchers from missing or oversimplifying instances and processes [56], it also makes it difficult for others to repeat the process to obtain the same results [84]. Another aspect of confirmability relates to the taxonomy evaluation in Section 5.1. Future research is needed in order to further assess the taxonomy's value [56], [57]. Here, the detailed descriptions of the data collection and analytical procedures in Section 3 and the taxonomy evaluation criteria will support researchers on using the taxonomy and the TOPS characteristics.

Transferability. Another potential limitation is that this research was conducted within a single organization. As such, we do not claim the findings are transferable to all other settings. Neither do we claim the list of coordination mechanisms to be exhaustive. Other organizations may use other mechanisms, depending on their unique coordination needs. It is also possible that the focus on inter-team coordination may have blinded us to the influence of team-level mechanisms and practices. However, the categories in the taxonomy and the TOPS characteristics are theoretically generalizable [65], because they are likely to be found also in other large-scale agile organizations [57]. However, different specific mechanisms may be identified from the literature and from other empirical settings [56], [57].

7 Conclusions

In this study, we addressed the research question, "Which inter-team coordination mechanisms are used in large-scale agile software development, and how do these mechanisms support inter-team coordination?" This is among the top concerns for researchers and practitioners in large-scale agile. We have analyzed data from 113 hours of observation and 31 interviews from a large-scale agile organization. From our findings, we make three contributions to the literature on coordination in large-scale software development.

First, we propose a taxonomy of inter-team coordination mechanisms with a total of 27 coordination mechanisms across three categories: Meetings, roles, tools, and artefacts. Second, we propose four key characteristics of coordination mechanisms that display a combination of social, technical, organizational, and physical characteristics. This resulted in the TOPS framework, which represents a novel approach to categorizing coordination mechanisms inspired from ideas of software engineering as a socio-technical practice. The framework builds on and extends previous research on coordination in agile software development. Third, we have provided an actionable approach to using the TOPS framework by introducing a visual template that can guide the practical mapping of inter-team coordination practices.

With these contributions we hope to advance knowledge on inter-team coordination in large-scale agile software development, and to support practitioners with coordination in our volatile, uncertain, and ever-changing contemporary business environments. The taxonomy and the TOPS framework are flexible approaches to inter-team coordination that take into account that coordination needs are changing. New mechanisms may easily be added as new coordination needs arise and new agile practices form. We encourage future research to use the taxonomy and the framework to provide rich descriptions of how coordination mechanisms are used to support inter-team coordination in large-scale agile.

Further, the TOPS framework may support researchers in tracking coordination changes over time by reassessing the mechanism's key characteristics at regular intervals. Future research should apply the TOPS framework in other large-scale settings to validate our findings in other large-scale settings. Finally, we believe it is important to recognize that a static view of coordination mechanisms may lead us to miss important insights. We therefore encourage future research on not only the change in using coordination mechanisms, but also on their changing characteristics in response to changing work conditions.

Acknowledgments

The authors wish to thank Entur and the informants for their willingness to share their experiences. We also thank the three anonymous reviewers for their valuable comments to earlier drafts of this paper. This research was supported in part by the Research Council of Norway through the research project Autonomous teams (A-teams) project, Grant Number 267704.

References

- [1] K. Dikert, M. Paasivaara, and C. Lassenius, “Challenges and success factors for large-scale agile transformations: A systematic literature review,” *Journal of Systems and Software*, vol. 119, pp. 87–108, Sep. 2016.
- [2] H. Edison, X. Wang, and K. Conboy, “Comparing Methods for Large-Scale Agile Software Development: A Systematic Literature Review,” *IEEE Transactions on Software Engineering*, 2021.
- [3] M. Paasivaara, B. Behm, C. Lassenius, and M. Hallikainen, “Large-scale agile transformation at Ericsson: a case study,” *Empirical Software Engineering*, vol. 23, no. 5, pp. 2550–2596, 2018.
- [4] T. Dingsøy, D. Falessi, and K. Power, “Agile development at scale: the next frontier,” *IEEE Software*, vol. 36, no. 2, pp. 30–38, 2019.
- [5] T. Dingsøy, N. B. Moe, T. E. Fægri, and E. A. Seim, “Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation,” *Empirical Software Engineering*, pp. 1–31, 2017.
- [6] S. Bick, K. Spohrer, R. Hoda, A. Scheerer, and A. Heinzl, “Coordination challenges in large-scale software development: a case study of planning misalignment in hybrid settings,” *IEEE Transactions on Software Engineering*, vol. 44, no. 10, pp. 932–950, 2018.
- [7] D. Batra, W. Xia, D. E. VanderMeer, and K. Dutta, “Balancing agile and structured development approaches to successfully manage large distributed software projects: A case study from the cruise line industry.,” *CAIS*, vol. 27, p. 21, 2010.
- [8] T. W. Malone and K. Crowston, “The interdisciplinary study of coordination,” *ACM Computing Surveys (CSUR)*, vol. 26, no. 1, pp. 87–119, 1994.
- [9] M. Kalenda, P. Hyna, and B. Rossi, “Scaling agile in large organizations: Practices, challenges, and success factors,” *Journal of Software: Evolution and Process*, vol. 30, no. 10, p. e1954, 2018.
- [10] T. W. Malone et al., “Tools for inventing organizations: Toward a handbook of organizational processes,” *Management Science*, vol. 45, no. 3, pp. 425–443, 1999.
- [11] J. Howison, J. Rubleske, and K. Crowston, “Coordination Theory: A Ten-Year Retrospective,” in *Human-computer Interaction and Management Information Systems: Foundations*, Routledge, 2015, pp. 134–152.
- [12] K. Crowston and C. S. Osborn, “A coordination theory approach to process description and redesign,” in *Organizing business knowledge: The MIT process handbook*, T. W. Malone, K. Crowston, and G. A. Herman, Eds. MIT press, 2003.
- [13] D. E. Strode, “A dependency taxonomy for agile software development projects,” *Information Systems Frontiers*, vol. 18, no. 1, pp. 23–46, 2016.
- [14] M. Berntzen, V. Stray, and N. B. Moe, “Coordination Strategies: Managing Inter-team Coordination Challenges in Large-Scale Agile,” in *Agile Processes in Software Engineering and Extreme Programming*, Cham, 2021, pp. 140–156.
- [15] N. B. Moe, T. Dingsøy, and K. Rolland, “To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software development,” *International Journal of Information Systems and Project Management*, vol. 6, no. 3, pp. 45–59, 2018.
- [16] M. Cataldo and J. D. Herbsleb, “Coordination breakdowns and their impact on development productivity and software failures,” *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 343–360, 2012.
- [17] G. A. Okhuysen and B. A. Bechky, “10 coordination in organizations: An integrative perspective,” *Academy of Management annals*, vol. 3, no. 1, pp. 463–502, 2009.
- [18] K. Schmidt and C. Simonee, “Coordination mechanisms: Towards a conceptual foundation of CSCW systems design,” *Computer Supported Cooperative Work (CSCW)*, vol. 5, no. 2–3, pp. 155–200, 1996.
- [19] V. Stray, D. I. Sjøberg, and T. Dybå, “The daily stand-up meeting: A grounded theory study,” *Journal of Systems and Software*, vol. 114, pp. 101–124, 2016.
- [20] V. Stray and N. B. Moe, “Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and Slack,” *Journal of Systems and Software*, vol. 170, p. 110717, 2020.
- [21] M. Paasivaara and C. Lassenius, “Empower Your Agile Organization: Community-Based Decision Making in Large-Scale Agile Development at Ericsson,” *IEEE Software*, vol. 36, no. 2, pp. 64–69, 2019.

- [22] E. Wenger, R. A. McDermott, and W. Snyder, *Cultivating communities of practice: A guide to managing knowledge*. Harvard business press, 2002.
- [23] D. Smite, N. B. Moe, G. Levinta, and M. Floryan, “Spotify Guilds: How to Succeed With Knowledge Sharing in Large-Scale Agile Organizations,” *IEEE Software*, vol. 36, no. 2, pp. 51–57, 2019.
- [24] D. E. Strode, S. L. Huff, B. Hope, and S. Link, “Coordination in co-located agile software development projects,” *Journal of Systems and Software*, vol. 85, no. 6, pp. 1222–1238, Jun. 2012.
- [25] M. Berntzen, N. B. Moe, and V. Stray, “The Product Owner in Large-Scale Agile: An Empirical Study Through the Lens of Relational Coordination Theory,” in *Agile Processes in Software Engineering and Extreme Programming*, Cham, 2019, pp. 121–136.
- [26] V. Braun and V. Clarke, “Using thematic analysis in psychology,” *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.
- [27] V. Braun and V. Clarke, “Thematic analysis,” in *APA handbook of research methods in psychology, Vol 2: Research designs: Quantitative, qualitative, neuropsychological, and biological.*, Washington, DC, US: American Psychological Association, 2012, pp. 57–71.
- [28] M. Fowler and J. Highsmith, “The Agile Manifesto,” 2001. [Online]. Available: <http://agilemanifesto.org/>
- [29] D. K. Rigby, J. Sutherland, and A. Noble, “Agile at scale: How to go from a few team to hundreds,” *Harvard Business Review*, vol. 96, no. 3, pp. 88–96, 2018.
- [30] L. Williams and A. Cockburn, “Guest Editors’ Introduction: Agile Software Development: It’s about Feedback and Change,” *Computer*, vol. 36, no. 6, pp. 39–43, 2003.
- [31] Digital.ai, “15th Annual State of Agile Report (2020),” 2021. [Online]. Available: <https://digital.ai/resource-center/analyst-reports/state-of-agile-report>
- [32] A. H. Van de Ven, A. L. Delbecq, and R. Koenig Jr, “Determinants of coordination modes within organizations,” *American sociological review*, pp. 322–338, 1976.
- [33] J. D. Thompson, *Organizations in action: Social science bases of administrative theory*. McGraw-Hill, 1967.
- [34] J. A. Espinosa, F. J. Lerch, and R. E. Kraut, “Explicit versus implicit coordination mechanisms and task dependencies: One size does not fit all,” in *Team cognition: Understanding the factors that drive process and performance.*, Washington, DC, US: American Psychological Association, 2004, pp. 107–129.
- [35] P. A. Jarzabkowski, J. K. Lê, and M. S. Feldman, “Toward a Theory of Coordinating: Creating Coordinating Mechanisms in Practice,” *Organization Science*, vol. 23, no. 4, pp. 907–927, 2012.
- [36] J. H. Gittell, “Relational coordination: Coordinating work through relationships of shared goals, shared knowledge and mutual respect,” *Relational perspectives in organizational studies: A research companion*, pp. 74–94, 2006.
- [37] R. Rico, M. Sánchez-Manzanares, F. Gil, and C. Gibson, “Team Implicit Coordination Processes: A Team Knowledge-Based Approach,” *Academy of Management Review*, vol. 33, no. 1, pp. 163–184, 2008.
- [38] K. Lewis, “Measuring transactive memory systems in the field: Scale development and validation,” *Journal of Applied Psychology*, vol. 88, no. 4, pp. 587–604, 2003.
- [39] E. Salas, D. E. Sims, and C. S. Burke, “Is there a ‘big five’ in teamwork?,” *Small group research*, vol. 36, no. 5, pp. 555–599, 2005.
- [40] P. H. Carstensen and C. Sørensen, “From the social to the systematic,” *Computer Supported Cooperative Work (CSCW)*, vol. 5, no. 4, pp. 387–413, 1996.
- [41] R. Giuffrida and Y. Dittrich, “A conceptual framework to study the role of communication through social software for coordination in globally-distributed software teams,” *Information and Software Technology*, vol. 63, pp. 11–30, 2015.
- [42] I. R. McChesney and S. Gallagher, “Communication and co-ordination practices in software engineering projects,” *Information and Software Technology*, vol. 46, no. 7, pp. 473–489, 2004.
- [43] A. Zaitsev, U. Gal, and B. Tan, “Coordination artifacts in agile software development,” *Information and Organization*, vol. 30, no. 2, p. 100288, 2020.
- [44] R. Hoda, J. Noble, and S. Marshall, “Self-Organizing Roles on Agile Software Development Teams,” *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 422–444, Mar. 2013.
- [45] M. Govers and P. van Amelsvoort, “A socio-technical perspective on the design of IT architectures: The lowlands lens,” *Management Studies*, vol. 6, no. 3, pp. 177–187, 2018.

- [46] C. Yang, P. Liang, and P. Avgeriou, "A systematic mapping study on the combination of software architecture and agile development," *Journal of Systems and Software*, vol. 111, pp. 157–184, 2016.
- [47] A. Sablis, D. Smite, and N. Moe, "Team-external coordination in large-scale software development projects," *Journal of Software: Evolution and Process*, p. e2297, 2020.
- [48] T. Dingsøy, N. B. Moe, and E. A. Seim, "Coordinating Knowledge Work in Multi-Team Programs: Findings from a Large-Scale Agile Development Program," *Project Management Journal*, vol. 49, pp. 64–77, 2018.
- [49] C. Larman and B. Vodde, *Large-scale scrum: More with LeSS*. Addison-Wesley Professional, 2016.
- [50] D. Leffingwell, *SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises*. Addison-Wesley Professional, 2018.
- [51] P. Kettunen and M. Laanti, "Future software organizations – agile goals and roles," *European Journal of Futures Research*, vol. 5, no. 1, p. 16, Dec. 2017.
- [52] K. Conboy and N. Carroll, "Implementing large-scale agile frameworks: challenges and recommendations," *IEEE Software*, vol. 36, no. 2, pp. 44–50, 2019.
- [53] Y. Shastri, R. Hoda, and R. Amor, "The role of the project manager in agile software development projects," *Journal of Systems and Software*, vol. 173, p. 110871, Mar. 2021.
- [54] J. M. Bass, "How product owner teams scale agile methods to large distributed enterprises," *Empirical Software Engineering*, vol. 20, no. 6, pp. 1525–1557, 2015.
- [55] M. Petre, J. Buckley, L. Church, M.-A. Storey, and T. Zimmermann, "Behavioral science of software engineering," *IEEE Software*, vol. 37, no. 6, pp. 21–25, 2020.
- [56] P. Ralph, "Toward methodological guidelines for process theories and taxonomies in software engineering," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 712–735, 2018.
- [57] R. C. Nickerson, U. Varshney, and J. Muntermann, "A method for taxonomy development and its application in information systems," *European Journal of Information Systems*, vol. 22, no. 3, pp. 336–359, 2013.
- [58] T. Dingsøy, T. E. Fægri, and J. Itkonen, "What is large in large-scale? A taxonomy of scale for agile software development," in *International Conference on Product-Focused Software Process Improvement*, Springer, Cham, 2014, pp. 273–276.
- [59] R. Florea and V. Stray, "The skills that employers look for in software testers," *Software Quality Journal*, vol. 27, no. 4, pp. 1449–1479, Dec. 2019.
- [60] D. Šmite, C. Wohlin, Z. Galviņa, and R. Prikladnicki, "An empirically based terminology and taxonomy for global software engineering," *Empirical Software Engineering*, vol. 19, no. 1, pp. 105–153, Feb. 2014.
- [61] P. Ralph et al., "ACM SIGSOFT empirical standards," 2020.
- [62] G. Walsham, "Doing interpretive research," *European Journal of Information Systems*, vol. 15, no. 3, pp. 320–330, Jun. 2006, doi: 10.1057/palgrave.ejis.3000589.
- [63] W. Hussain et al., "How Can Human Values Be Addressed in Agile Methods A Case Study on SAFe," *IEEE Transactions on Software Engineering*, 2022.
- [64] H. Sharp, Y. Dittrich, and C. R. B. de Souza, "The Role of Ethnographic Studies in Empirical Software Engineering," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 786–804, Aug. 2016.
- [65] M. Crang and I. Cook, *Doing ethnographies*. Sage, 2007.
- [66] P. R. Niven and B. Lamorte, *Objectives and key results: Driving focus, alignment, and engagement with OKRs*. John Wiley & Sons, 2016.
- [67] Stray, Viktoria, Moe, Nils Brede, Vedal, Henrik, and Berntzen, Marthe, "Using Objectives and Key Results (OKRs) and Slack: A Case Study of Coordination in Large-Scale Distributed Agile," *Techrxiv*. Preprint, Nov. 2021, doi: 10.36227/techrxiv.16892161.v1.
- [68] J. M. Bass and A. Haxby, "Tailoring product ownership in large-scale agile projects: managing scale, distance, and governance," *IEEE Software*, vol. 36, no. 2, pp. 58–63, 2019.
- [69] T. Dybå, T. Dingsøy, and N. B. Moe, "Agile project management," in *Software project management in a changing world*, Springer, 2014, pp. 277–300.
- [70] M.-A. Storey, N. A. Ernst, C. Williams, and E. Kalliamvakou, "The who, what, how of software engineering research: a socio-technical framework," *Empirical Software Engineering*, vol. 25, no. 5, pp. 4097–4129, 2020.

- [71] R. Hoda, "Socio-Technical Grounded Theory for Software Engineering," *IEEE Transactions on Software Engineering*, 2021, doi: <https://doi.org/10.1109/TSE.2021.3106280>.
- [72] W. J. Orlikowski and S. V. Scott, "10 Sociomateriality: Challenging the Separation of Technology, Work and Organization," *Academy of Management Annals*, vol. 2, no. 1, pp. 433–474, 2008.
- [73] R. P. Bostrom and J. S. Heinen, "MIS problems and failures: A socio-technical perspective. Part I: The causes," *MIS quarterly*, pp. 17–32, 1977.
- [74] I. Bider, "Is People-Structure-Tasks-Technology Matrix Outdated?," in 3rd International Workshop on Socio-Technical Perspective in IS development (STPIS'17), CEUR-WS. org, 2017, pp. 90–97.
- [75] S. Alter, "Applying Socio-technical Thinking in the Competitive, Agile, Lean, Data-Driven World of Knowledge Work and Smart, Service-Oriented, Customer-Centric Value Creation Ecosystems," *Complex Systems Informatics and Modeling Quarterly*, no. 18, pp. 1–22, 2019.
- [76] H. Kim, D.-H. Shin, and D. Lee, "A socio-technical analysis of software policy in Korea: Towards a central role for building ICT ecosystems," *Telecommunications Policy*, vol. 39, no. 11, pp. 944–956, 2015.
- [77] P. Ralph et al., "Pandemic programming," *Empirical Software Engineering*, vol. 25, no. 6, pp. 4927–4961, 2020.
- [78] D. Smite, M. Mikalsen, N. B. Moe, V. Stray, and E. Klotins, "From Collaboration to Solitude and Back: Remote Pair Programming During COVID-19," in *International Conference on Agile Software Development*, Springer, Cham, 2021, pp. 3–18.
- [79] D. Russo, P. H. Hanel, S. Altnickel, and N. van Berkel, "Predictors of well-being and productivity among software professionals during the COVID-19 pandemic—a longitudinal study," *Empirical Software Engineering*, vol. 26, no. 4, pp. 1–63, 2021.
- [80] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," in *Future of Software Engineering (FOSE'07)*, IEEE, 2007, pp. 188–198.
- [81] J. A. Espinosa, S. A. Slaughter, R. E. Kraut, and J. D. Herbsleb, "Team knowledge and coordination in geographically distributed software development," *Journal of management information systems*, vol. 24, no. 1, pp. 135–169, 2007.
- [82] M. Berntzen and S. I. Wong, "Autonomous but Interdependent: The Roles of Initiated and Received Task Interdependence in Distributed Team Coordination," *International Journal of Electronic Commerce*, vol. 25, no. 1, 2021.
- [83] L. Fosslien and M. W. Duffy, "How to combat zoom fatigue," *Harvard Business Review*, vol. 29, 2020.
- [84] E. G. Guba, "Criteria for assessing the trustworthiness of naturalistic inquiries," *Ectj*, vol. 29, no. 2, pp. 75–91, 1981.
- [85] C. Robson and K. McCartan, *Real world research: a resource for users of social research methods in applied settings*. Wiley, 2016.
- [86] D. Russo, "The Agile Success Model: A Mixed-methods Study of a Large-scale Agile Transformation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 4, pp. 1–46, 2021.



Marthe Berntzen is a Ph.D. candidate in Software Engineering at the department of Informatics, University of Oslo. She received the M.Sc. degree from BI Norwegian Business School and has four years of industry experience. Marthe's Ph.D. research centers around inter-team coordination in large-scale agile software development. Her research interests include agile methods and practices, coordination of teamwork, leadership, and coordination in distributed settings. Her work has been presented at international conferences within software engineering, information systems and management and she has published research on leadership and coordination.



Rashina Hoda is an Associate Professor of Software Engineering at Monash University, Melbourne. Rashina specializes in human-centered software engineering, including agile transformations, self-organizing teams, agile project management, and large-scale agile, and has introduced socio-technical grounded theory to software engineering. She serves as an Associate Editor of the IEEE Transactions on Software Engineering and as co-chair for ICSE-SEIS 2023, and previously, on the advisory board of IEEE Software and as PC co-Chair for CHASE2021. For more: www.rashina.com.



Nils Brede Moe is a chief scientist at SINTEF. He works with software process improvement, intellectual capital, autonomous teams, and agile and global software development. He has led several nationally funded software engineering research projects covering organizational, sociotechnical, and global/distributed aspects. Moe received a Dr.Philos. in Computer Science from the Norwegian University of Science and Technology and holds an adjunct position at the Blekinge Institute of Technology in Sweden.



Viktoria Stray is an Associate Professor at the University of Oslo's Department of Informatics. She also holds a research position at SINTEF. Her research interests include agile methods, coordination, global software engineering, software testing, and large-scale development. The main focus of her research is to improve the productivity of developers and testers and increase the success of software projects. Stray has a Ph.D. in Software Engineering and has worked several years in the industry participating in some of the largest software development projects in Norway.

Paper 4: Responding to Change Over Time: A Longitudinal Case Study on Changes in Coordination Mechanisms in Large-Scale Agile

Marthe Berntzen, Viktoria Stray, Nils Brede Moe and Rashina Hoda

Accepted for publication in Empirical Software Engineering

Abstract

Context: Responding to change and continuously improving processes, practices, and products are core to agile software development. It is no different in large-scale agile, where multiple software development teams need to respond both to changes in their external environments and to changes within the organization.

Objective: With this study, we aim to advance knowledge on coordination in large-scale agile by developing a model of the types of organizational changes that influence coordination mechanisms.

Method: We conducted a longitudinal case study in a growing large-scale agile organization, focusing on how external and internal changes impact coordination over time. We collected our data through 62 days of fieldwork across one and a half years. We conducted 37 interviews, observed 118 meetings at all organizational levels, collected supplementary material such as chat logs and presentations, and analyzed the data using thematic analysis.

Results: Our findings demonstrate how external events, such as onboarding new clients, and internal events, such as changes in the team organization, influence coordination mechanisms in the large-scale software development program. We find that external and internal change events lead to the introduction of new coordination mechanisms, or the adjustment of existing ones. Further, we find that continuous scaling requires continuous change and adjustment. Finally, we find that having the right mechanisms in place at the right time strengthens resilience and the ability to cope with change in coordination needs in complex large-scale environments.

Conclusions: Our findings are summarized in an empirically based model that provides a practical approach to analyzing change, aimed at supporting both researchers and practitioners dealing with change in coordination mechanisms in large-scale agile development contexts.

Keywords: Large-scale agile, software development, coordination, organizational change, continuous improvement, longitudinal case study

1 Introduction

Agile software development welcomes change (Fowler and Highsmith 2001), and large-scale agile is abundant with changes, for example, in customer requirements, technical dependencies, team composition, and tool use. In large-scale agile, defined as software development involving more than six teams or more than 50 developers¹ (Dikert et al. 2016), multiple teams join efforts in developing an overall software system. Large-scale organizations must cope with rapid external disruptions, such as technological innovation, economic and political destabilization, and re-negotiation of workplace arrangements while continuously improving their software engineering practices. Additionally, dependencies between teams represent further challenges to efficient software delivery. Therefore, understanding how change impacts coordination may make the difference between successful and non-successful software development in a large-scale context.

Coordination, often defined as the management of dependencies (Malone and Crowston 1994), is central to agile software development because of dependencies that may impact software delivery efficiency (Strode 2016). Dependencies occur "when the progress of one action relies on the timely output of a previous action or on the presence of a specific thing, where a thing can be an artifact, a person, or a piece of information" that "can be managed well, poorly, or not at all" (Strode 2016, p. 24). Coordination is needed because if dependencies are insufficiently managed, they can cause blockages and bottlenecks that delay the development progress and, ultimately, software delivery (Cataldo and Herbsleb 2012).

When multiple teams work together to develop software, several coordination challenges arise (Dingsøy et al. 2017). For example, interfacing between teams becomes problematic because dependencies in one team may delay or hinder the work of other teams (Bick et al. 2018), and achieving and maintaining technical consistency becomes difficult (Dikert et al. 2016). Such challenges require that agile practices are adapted to the large-scale level (Dingsøy et al. 2017). However, the self-organizing teams model central to agile can become problematic because of the need to align and coordinate interdependent teams' work practices and outputs (Moe et al. 2021). These challenges must be solved within an ever-changing context, making it essential to use the most effective *coordination mechanisms*, which we define as organizational processes, entities, or arrangements used to manage dependencies between activities to realize a collective performance (Okhuysen and Bechky 2009). Selecting the most suitable mechanisms at any given time and modifying or replacing them in response to change is also a challenge.

¹ There is no agreed-upon definition of what exactly constitutes 'large-scale' in the research community (Edison et al. 2022). In line with Dikert et al. (2016), we define large-scale agile as more than six teams or involving more than 50 developers. Our case organization, Entur, eventually had 17 teams and could, therefore, also have been classified as 'very large-scale' according to a much-used definition (Dingsøy et al. 2014). However, because the size of the program over time grew from 'large-scale' to 'very large-scale' (Dingsøy et al. 2014), we keep with the general term 'large-scale' to better relate to the literature on large-scale agile overall (Edison et al. 2022; Uludağ et al. 2022).

Change in coordination is a topic in need of further exploration, especially in the context of large-scale agile, because changes bring new and different coordination requirements that, if insufficiently managed, can cause delays and bottlenecks, and even project breakdown (Cataldo and Herbsleb 2012; Dikert et al. 2016; Bick et al. 2018). In this study, we consider two forms of change relevant to coordination in large-scale agile. On the one hand, change can be understood as *event-based*, that is, as "something specific that happens" through disruptive events or patterns of events (Jarzabkowski et al. 2012). Software engineering studies have focused on 'a change' either by studying the 'before and after' a large-scale agile transformation or while following what happens during the implementation of a large-scale agile framework (e.g., Paasivaara et al. 2018; Russo 2021; Gustavsson et al. 2022). On the other hand, change can be understood as a *continuous process* or flow of activities that are harder to pinpoint but easily observable in retrospect (Van de Ven and Poole 2005; Langley et al. 2013). We believe that both perspectives of change can inform the analysis of how organizational changes influence coordination mechanisms and how the mechanisms themselves change over time.

As research on coordination in agile software development and large-scale agile is maturing (Dingsøyr et al. 2012; Hoda et al. 2018; Berntzen et al. 2022), we believe it is important to focus on understanding how and why coordination practices and mechanisms change over time. In this paper, we aim to explore the relationship between organizational changes and changes in coordination mechanisms in the context of large-scale agile software development. We do this by investigating the following research question:

What type of organizational changes influence coordination mechanisms in large-scale agile, and how do these mechanisms change over time?

We report on findings from a longitudinal case study conducted over one and a half years in a large-scale organization called Entur. The time frame enabled us to follow the case organization as changes were taking place. We spent a total of 62 days at the field site, and observed 108 meetings, such as retrospective and stand-up meetings, client meetings, and board meetings, conducted 37 interviews with practitioners in roles such as team leaders, product owners, and program managers, and collected supplementary material from resources, such as chat logs and documentation pages as the development program continued to scale. The data were analyzed using thematic analysis (Braun and Clarke 2006, 2012) in light of the theoretical framework of Jarzabkowski et al. (2012), who proposed a process theory of how coordination mechanisms are created in light of organizational change. As longitudinal studies of such scope and detail are rare within software engineering (Sharp et al. 2016) this study represents a unique contribution to the literature on large-scale agile coordination over time. Further, in comparison to other studies dealing with change in coordination mechanisms in large-scale agile (Moe et al. 2018; Gustavsson 2019; Dingsøyr et al. 2022), this study also focuses on *change* itself as opposed to the result of a change.

Our study offers the following contributions to software engineering research and practice:

- We advance research on change in coordination mechanisms over time in large-scale agile software development, which is called for in previous works (Moe et al. 2018; Dingsøy et al. 2022).
- We provide a rich empirical description with a unique level of depth in the data collection.
- We build on existing theoretical work to propose a model of change in coordination mechanisms in large-scale agile (Figure 7).
- We provide an actionable approach to analyzing change for practitioners who want to deep-dive into understanding and responding to change in coordination mechanisms (Table 4).

The paper is organized as follows. In Section 2, we review some of the existing research on coordination and change in large-scale agile. Section 3 presents the case organization and provides detailed information about our research methodology. Section 4 presents our research findings, which are discussed in Section 5, where we also discuss the practical and theoretical implications of the study. Section 6 concludes the paper.

2 Background

In presenting the background literature informing our study, we first consider different approaches to change in large-scale agile. Next, we present background on coordination and coordination mechanisms before presenting a process theoretical approach to studying change in coordination mechanisms.

2.1 Change in Large-Scale Agile Software Development

Change is central to agile software development. “Responding to change” is part of the agile manifesto (Fowler and Highsmith 2001) and core to agile. Traditionally, studies have focused on how software development teams respond to change (Hoda and Noble 2017; Spiegler et al. 2021). Further, changes are typically understood as “things that happen,” that is, events that are more or less beyond the individual developer’s or team’s control, for example, changes in requirements (Aldave et al. 2019; Madampe et al. 2022), re-organizing of the team (Spiegler et al. 2021) or company structure (Gustavsson et al. 2022; Carroll et al. 2023), or technical issues (Kwan et al. 2011; Cataldo and Herbsleb 2012). In large-scale agile, such events can be even more complex and challenging because of the dependencies that exist between teams that develop an overall product with inter-dependent components or several inter-dependent products.

The notion of *continuous improvement* is also core to agile because agile teams constantly strive to find better ways of solving their day-to-day challenges (Fitzgerald and Stol 2017). In large-scale settings, where inter-team coordination is needed, continuous improvement of coordination practices is vital for teams to successfully keep up with each other (Kalenda et al. 2018; Paasivaara et al. 2018; Dingsøy et al. 2022). Using an efficient mix of coordination mechanisms appears essential to respond to changes and continuously improve (Strode 2016). However, what constitutes an optimal mix of mechanisms may change over time (Moe et al. 2018; Dingsøy et al. 2022). This is particularly true in uncertain situations, such as

technological organizations, where hierarchies and rules-based systems are less useful (Jarzabkowski et al. 2012). Continuous improvement means continuously making changes. Accordingly, change can be understood as a continuous flow of activities. From this perspective, it is difficult to pinpoint precisely when such changes occur, but they are easily observable retrospectively (Van de Ven and Poole 2005; Langley et al. 2013).

As research and practice are maturing, different terms are used to describe variations of large-scale agile approaches. A recent study separates first- and second-generation large-scale agile development methods (Dingsøyr et al. 2022). First-generation methods are described as development methods that combine agile and traditional methods, typically using agile at the team level and traditional project management practices used at the inter-team, project, or organizational level (e.g., Batra et al. 2010; Bick et al. 2018). Many organizations use this type of mix between agile practices and other project management practices because large-scale organizations (need to) have other governance structures surrounding the development and other agile business activities (Kalenda et al. 2018; Edison et al. 2022). Second-generation methods use ideas from the agile and lean communities, focusing on the product, collaboration, informal communication, and flexible and evolutionary delivery models and organizations (Dingsøyr et al. 2022). The commercial large-scale agile development frameworks, including SAFe, Large-Scale Scrum (LeSS), and the Spotify model, are examples of second-generation agile development methods (Dingsøyr et al. 2022). These and other scaling frameworks are widely used by practitioners in large-scale agile because they propose specific processes and mechanisms to manage the challenges with dependencies in large-scale software development (Dikert et al. 2016; Edison et al. 2022). Much existing research on large-scale agile transformation, including the studies referenced in the coming section, are case studies of companies implementing one of the large-scale agile frameworks.

However, not all large-scale development projects and programs use large-scale frameworks or methods. Some research critiques against large-scale frameworks are that they are not flexible enough to handle changing coordination needs (Gustavsson et al. 2022). In practice, many organizations take a less rigorous methodological approach to large-scale development. The comprehensive literature review by Edison et al. (2022) shows that most organizations adapt agile methods to fit their specific contextual needs, regardless of adopting a large-scale framework. Irrespective of the approach taken, both researchers and practitioners agree that coordination is a crucial challenge to the success of large-scale agile development (Dikert et al. 2016; Edison et al. 2022; Uludağ et al. 2022).

2.2 Coordination and Coordination Mechanisms in Large-Scale Agile

Coordination has been studied across a wide range of research fields, including management, organization studies, information systems management, and software engineering (Espinosa et al. 2007; Okhuysen and Bechky 2009). Early studies on management and organization (e.g., March and Simon 1966; Thompson 1967) recognized the need to coordinate interdependent processes and activities, and the topic is still widely studied today (Castañer and Oliveira 2020). Within large-scale agile, research on coordination has dealt specifically with dependency management, as this represents a key challenge to the success of large-scale development (Stray et al. 2022b, a). An analysis of agile teams resulted in three dependency groups: knowledge,

resource, and process dependencies (Strode 2016). *Knowledge dependencies* relate to information required for progress, including historical knowledge and task allocation knowledge. *Process dependencies* include activities and business processes that must be in place for tasks to proceed, while *resource dependencies* include technical dependencies and dependencies related to the availability of some resource (i.e., a person, a place, or a thing) (Strode 2016). These dependency categories form a taxonomy of dependencies for agile teams, which has also been applied at the large-scale level by focusing on dependencies between teams (Berntzen et al. 2021).

Dependencies are managed using *coordination mechanisms*, which we define as organizational processes, entities, or arrangements used to manage dependencies between activities to realize a collective performance (Okhuysen and Bechky 2009). Coordination mechanisms have been operationalized differently across fields. Early conceptualizations include work standardization, outputs, skills, and norms as central coordination mechanisms (Mintzberg 1989). Malone and Crowston (1994) propose coordination mechanisms such as priority orders, budgets, sequencing, tracking, and standardization. Another approach is the three modes of coordination mechanisms introduced by Van de Ven and colleagues (1976), which includes a group mode (i.e., mechanisms based on mutual adjustment through feedback), a personal mode (i.e., mutual adjustment based on feedback between two people), and the impersonal mode (i.e., the use of plans, schedules, and standardized information). Much research on agile development and coordination in software engineering relies on Van de Ven's (1976) coordination modes and Malone and Crowston's (1994) coordination definition. In addition, Strode and colleagues (2012) have developed a theory of coordination in agile development teams, which has recently been applied at the inter-team level (Berntzen et al. 2021; Stray et al. 2022a).

In large-scale agile, teams working on different parts of the overall system need to coordinate, for example, merging of code, testing, and releases. In such situations, teams rely on mechanisms adapted to be used across teams, which we refer to as *inter-team coordination mechanisms*. In recent work, we presented a taxonomy of inter-team coordination mechanisms specific to large-scale agile (Berntzen et al. 2022) consisting of meetings, such as stand-up meetings, retrospectives, and communities of practice (Moe et al. 2018); roles, such as product owners (Bass 2015); and tools and artifacts, including communication and documentation platforms, such as Slack, JIRA, and Confluence (Lin et al. 2016), and goal management tools such as Objectives and Key Results (OKRs) (Niven and Lamorte 2016; Stray et al. 2022b). As changes in the development process, such as changing requirements or change in tool use, team or role composition, or system architecture, for example, are likely to have a multi-team impact with often far-reaching consequences (Cataldo and Herbsleb 2012; Dikert et al. 2016), understanding change in relation to coordination is of great importance.

There are few studies of change in coordination in large-scale agile, but the topic is gaining traction. For example, Gustavsson and colleagues (Gustavsson 2019; Gustavsson et al. 2022) investigated how inter-team coordination and team autonomy change when a large-scale framework (such as SAFe) is implemented. Moe et al. (2018) investigated how meetings changed over time, using Van de Ven's (1976) classification of coordination meetings, i.e., the group mode. Their study showed that scheduled meetings were important coordination mechanisms early in the development process but that more unscheduled meetings could

replace these arenas over time. Similar findings were reported by Dingsøy et al. (2018), who, in addition to the change in meeting formats, found more use of horizontal coordination and change in tool use early versus late in the development program as the case organization matured. Paasivaara et al. (2018) found that in addition to a change in tools and practices (i.e., coordination mechanisms), a shift in mindset was important when conducting a large-scale agile transformation. In a recent study, Dingsøy et al. (2022) describe changes in inter-team coordination in a large-scale development program that transitioned from using a combination of traditional and agile development practices (i.e., first-generation development methods) to using cross-functional autonomous teams and continuous delivery (i.e., second-generation development methods). Their research showed that the number of coordination mechanisms went down from 27 to 14 when the program started using a second-generation development method and that coordination effectiveness was perceived as higher by interview participants.

Common to these previous studies is the focus on *one specific change* (i.e., how was coordination before/after implementing a framework (Gustavsson et al. 2022; Carroll et al. 2023) or agile transformation (Paasivaara et al. 2018), transitioning from one phase to another (Dingsøy et al. 2022) or focus on one type of coordination mechanism (such as group mode coordination mechanisms (Dingsøy et al. 2018; Moe et al. 2018)).

2.3. A Process-theoretical Approach to Change in Large-scale Agile

In this study, we focus on the phenomenon of change in relation to coordination in large-scale agile, using a process-theoretical lens. Organizational researchers have studied change for decades, often using process theories to capture the complexities of change and evolution over time (Pettigrew 1990; Langley 1999; Van de Ven and Poole 2005). A process theory aims at explaining and understanding how an entity changes and develops over time (Langley et al. 2013). Process theories are suitable for dealing with change and with time (Van de Ven and Poole 2005) through their ability to explain the temporal order of events based on historical narratives (Gregor 2006; Langley et al. 2013) and have been applied in research on software development and on coordination in several ways to explain how changes in organizations may unfold. For instance, Allison and Merali (2007) proposed a theory of software process improvement, where the interplay between software development and software process improvement continuously informed each other and where both process and product were changing each other, and were changed by their surrounding context over time. In a recent paper, Carroll et al. (2023) applied normalization process theory (Murray et al. 2010) to examine how agile practices were embedded and sustained in a large international company that implemented the Spotify model during a large-scale agile transformation. They found that a failure to normalize new practices led to the unraveling of the transformation within 18 months.

In a longitudinal case study of coordination in a large-scale technology company that underwent an organizational restructuring, Jarzabkowski et al. (2012) proposed a process-theoretical framework to explain how coordination mechanisms are created in practice through five cycles. The process starts with some disruptive event, such as a reorganization, restructuring or transformation, that *disrupts existing ways of coordinating*. In the second cycle, actors are trying and failing to coordinate effectively and thereby *orients to absences in*

coordinating. Third, new efforts to coordinate are made to fill the absences, which *creates new elements of coordinating*. In the fourth cycle, *new patterns of coordinating* are formed as links are created between elements of the new coordination mechanism. In the fifth and final cycle, the new coordination *patterns are stabilized* as new mechanisms are formalized (Jarzabkowski et al. 2012).

In our analysis and results, presented in the coming sections, we draw on the theoretical framework by Jarzabkowski et al. (2012). This framework is suitable because it explicitly recognizes that coordination mechanisms are not stable entities but are created over time in response to changes (i.e., disruptions). Such a view of coordination mechanisms appears highly compatible with large-scale agile software development, where responding to change is vital to succeeding. However, this study focuses specifically on disruptive events, which we understand as events of a certain magnitude, such as shutting down organizational departments, mergers and acquisitions, or changing an organization's technological platform (Jarzabkowski et al. 2012). Additionally, similar to the examples from large-scale agile literature cited in the previous section, Jarzabkowski et al. (2012) also focused on *one specific* event. Because continuous improvement is core to agile, we wanted to capture how the many events, large and small, as well as the more subtle, ongoing changes, shape coordination practices over time. Moreover, Jarzabkowski et al. (2012) limit their discussion to how coordination mechanisms are created. Because changes are omnipresent in large-scale software development, we wanted to understand not only how mechanisms are created but also how they change in response to internal and external changes, and whether the five-cyclical model can also apply to ongoing changes in coordination mechanisms. To this end, we also draw on the notion of continuous improvement (Fitzgerald and Stol 2017) and an understanding of change as a continuous process of activities (Langley and Truax 1994; Langley 1999) introduced in the above sections.

3 Research Methods

In this study, we explore what types of organizational changes influence coordination mechanisms over time through a case study conducted over 1.5 years in a large-scale agile program in an organization called Entur. The case study approach allowed for a deep, situational understanding of the research topic (Flyvbjerg 2006), and the longitudinal format provided the opportunity to investigate research questions of temporal character ((Van de Ven and Poole 2005; Langley et al. 2013). In software engineering, case studies are valuable for understanding software development activities in context (Runeson and Höst 2008; Wohlin and Aurum 2015). We chose a single case to investigate our research question because we had the opportunity to conduct longitudinal fieldwork in an interesting organization where we were given access to many and varied data sources, as described below. Single-case studies should be motivated by opportunities for learning about a phenomenon of interest (Flyvbjerg 2006), and the case needs to be relevant enough to provide such opportunities. Flyvbjerg (2006) defines a *critical case* as a case that has “strategic importance in relation to a general problem” (p. 229), in our case, coordination in large-scale agile software development. Entur can be considered a critical case because of their size and number of teams with varying levels of interdependence, making them likely to experience many of the coordination challenges outlined in Section 2. The features

described in the following are, however, not unique to this organization. Therefore, it can be argued that what is valid (or not) for this case would also be valid (or not) for many cases (Flyvbjerg 2006). Single-case studies are further suitable for process-theoretical research because of the level of detail required for understanding the intricacies of processual change in organizations that are not easily captured using formal variance theories with higher levels of abstraction (Van de Ven and Poole 2005; Ralph 2018).

In the following, we rely on two sensemaking strategies to present the case and our findings. We use a narrative strategy (Pettigrew 1990) to convey the rich textual information and to highlight the identified change themes. Additionally, we rely on the visual mapping strategy (Langley and Truax 1994; Langley 1999), as presented in Figure 4, to visually represent the findings.

3.1 Case Description

Our case, Entur, is a public sector IT organization established in 2016 in response to a public transportation reform initiated by the Norwegian Ministry of Public Transportation. Entur aims to make public transportation an easier and more viable option for the Norwegian public. To fill this mandate, Entur develops several products related to public transport in Norway. One central product is a multi-platform travel planner that aims to allow travelers to plan and manage their entire trip within one single application. Another is a new sales platform and API that railway operators and other public transportation operators can use to distribute their products to the public through the Entur app or their own channels. Finally, they collect, refine, and share public transportation data through open APIs². Because they used agile methods right from the beginning, Entur can be considered a mature agile development program. They have also experienced substantial organizational growth within few years and have, at the same time, been successful in delivering on the goals of the public transportation reform. The company is recognized within its national context, for example, by the Digitalization Council of Norway³.

The large-scale agile team environment. Entur started working with agile methods from day one and, therefore, never underwent an agile transformation as part of their organizational scaling journey. Since the outset in 2016, the development organization has grown rapidly, from five teams in 2016 to 17 in January 2020, and the growth has continued. Figure 1 displays the team organization in 2018 when we started our fieldwork. Despite being a large-scale agile organization from the outset, Entur has chosen not to adopt any scaling framework (such as SAFe or LeSS). Instead, they use software development best practices and gain inspiration from companies such as Spotify and Google, and tailor any new approach to their specific needs. For example, they established a biweekly tech lead forum modeled from Spotify's guilds and experimented with using OKRs (used and popularized by Google from around 2016 (Niven and Lamorte 2016)).

² See www.entur.org for more information.

³ www.digdir.no.

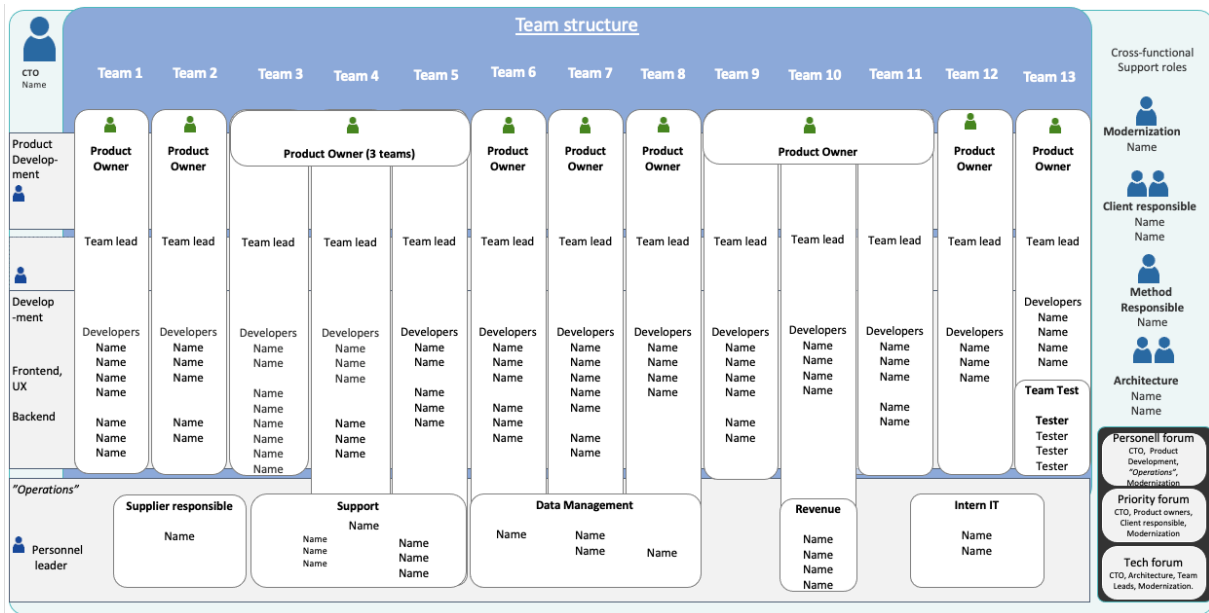


Figure 1. The team organization of the development program in 2018.

The development teams were cross-functional but focused on different parts of the overall deliveries, such as sales, ticketing, pricing, and web and app. Team size and composition varied slightly over time; the largest team had more than 16 members, while the smallest had about five. All teams had a team leader and a product owner, and a tech lead after this role emerged during 2019. All additional team members were developers that in principle could work on any part of their teams' code through shared repositories and documentation. In addition, a dedicated test team worked with the different teams to ensure appropriate testing of the different parts of the system. Some teams, such as the sales and product teams worked primarily with backend, whereas web and app teams worked with front-end and UX. Because all teams were connected at some level in delivering the overall product, all teams needed to coordinate. However, some teams had more dependencies to other teams than others. For example, almost all other teams had dependencies to the team that wrote the tickets and on-board products, and the UX and web team were dependent on almost all other teams, in that they needed the output of other teams to be ready in order to deliver visual output to travelers using the Entur app. The teams were autonomous in the choice of development methods, and most teams used practices from Scrum and Kanban. As such, there was a culture of testing new approaches, both within teams and at the inter-team level. At the same time, the need for inter-team coordination and some level of alignment across teams was high as they grew from five to 17 teams. The size and complexity of both the technology and the large-scale organization came with the consequence of many and complex knowledge, technology, and process dependencies. As such, coordination was an ongoing and evolving challenge. More details on the evolution and scaling of the organization and the coordination practices and mechanisms are provided in Section 4.

3.2 Data Collection

Data was collected from August 2018 to January 2020. During the one and a half years of fieldwork, the first author conducted observations and interviews and collected supplementary material such as meeting minutes, Slack (a collaborative communication tool) logs, e-mails,

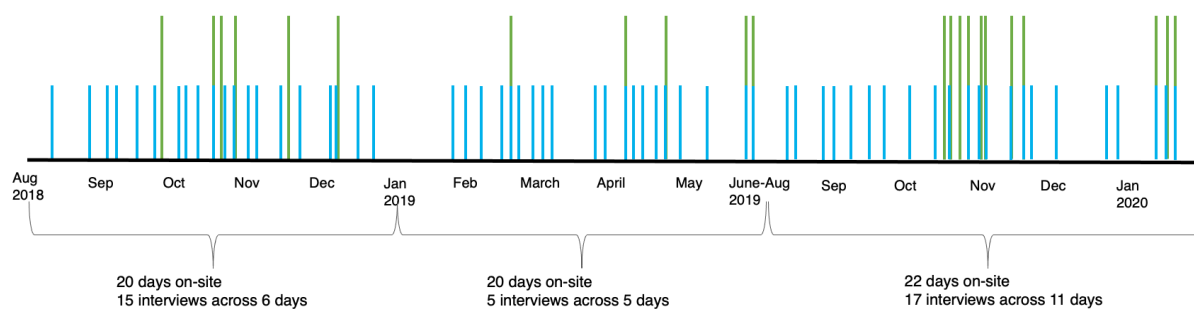


Figure 2. Data collection timeline. Short, blue bars represent unique observation days on-site, while long, green lines indicate that interviews were conducted that day.

and Confluence (a software documentation tool) pages. The second and third authors participated in some of the data collection. Figure 2 displays a timeline of the data collection. Data collection ended in January 2020, when the pre-planned fieldwork period of the first author had come to an end. We chose an ethnographic approach to data collection (Sharp et al. 2016), which includes researcher immersion in the case context and longer periods of fieldwork with detailed observation and extensive notetaking following an observational protocol (Crang and Cook 2007). In software engineering, ethnographic approaches provide opportunities for a detailed understanding of the development practice, including both social and technical aspects of the development process (Sharp et al. 2016). Our ethnographic approach included long-term non-participant observation, conducted in face-to-face settings, undertaken to understand and capture coordination in large-scale agile, and conducted with process theoretical underpinnings (Sharp et al., 2016).

This study extends our previous research, and parts of the data material have been analyzed for other studies. Specifically, we analyzed 12 interviews and observations from 17 meetings from September – November 2018 in Berntzen et al. (2019). In another study (Berntzen et al. 2021), we used data from August 2019 to January 2020, including 12 interviews and observations from 26 meetings. Finally, 31 interviews, observation notes from 94 meetings, and supplemental material such as Slack logs were analyzed in developing the taxonomy of inter-team coordination mechanisms and the TOPS framework (Berntzen et al. 2022). The study presented in this paper adds to previous studies with new analyses that shed light on previously unreported change-related processes and events, with a unique focus on studying them over time. In approaching the research question for this study, new data was added, including the full range of observation notes from 62 days on-site and 14 new meetings, including six client meetings, five ‘change workshops’ where organizational and structural changes were discussed with internal and external representatives and three other meetings including two external client preparation meetings and a board meeting. We also included six additional interviews conducted with Entur’s agile methods specialist. While confidentiality clauses prevent us from sharing original data material, we share examples from all data sources throughout the manuscript. Table 1 provides details of the underlying data material supporting the study.

Observations. Observations were conducted on a regular basis, as shown by the short bars in Figure 2. The first author conducted all observations. Additionally, the third author was present on a few occasions, for example, during the ‘change workshops’. Because we wanted a broad

Table 1. Data collection details by type of data material

Observations	Type of observation	Number
	Internal meetings:	
	Prioritization meetings	10
	Tech lead forum (Community of practice meeting)	7
	Weekly program demos	7
	Product owner weekly meetings	6
	Inter-team stand-up meetings	6
	Inter-team retrospectives	4
	OKR workshops	2
	Ad hoc inter-team meetings	26
	Intra-team meetings	26
	External meetings:	
	'Change workshop'	5
	Client meetings	6
	Other meetings	3
	Total number of meetings observed	108
	Unique days on-site	62
Interviews	Roles interviewed (Gender/Mean tenure IT/Mean tenure company)	
	Product owners (5 male, 4 female, IT tenure 11.5 years, company 1.8 years)	9
	Program managers (4 male, 1 female, IT tenure 18 years, company tenure 1.6 years)	5*
	Program architects (4 male, IT tenure 19 years, company tenure 1.4 years)	4*
	Tech leads (3 male, 1 female, IT tenure 7 years, company tenure 2.4 years)	4
	Team leaders (2 male, IT tenure 9 years, company tenure 1.5 years)	2
	Agile methods specialist (male, IT tenure 15 years, company tenure 4 years)	1**
	Unique individuals	25
	*Six participants were interviewed twice or more (4 managers and 2 architects)	6
	**Recurring interviews with the agile methods specialist	6
	Total number of interviews	37
Supplementary documentation	Slack logs, JIRA and Confluence documentation, e-mails, internal and external documents (e.g., presentations, reports, minutes).	

and detailed data material, we observed both meetings and the everyday work at the office. The observation days varied somewhat across the weekdays during the 1.5 years. Our presence varied as we wanted to observe the broad range of inter-team meetings conducted across the week. For example, we could be present one week on Monday and Thursday, the next on a Wednesday, and yet another week we could be absent. For the meeting observations, we observed meetings on all organizational levels, primarily inter-team meetings such as the product owner prioritization meeting and the tech lead forum, and team-level meetings such as team retrospectives and daily stand-up meetings. We were also able to observe client meetings and a board meeting. Because of our ongoing presence, we were able to join in on spontaneous ad hoc meetings as well as planned meetings. We used an observation protocol detailing, for example, the physical setting, people present, and tools and artifacts used. These observations left us with a rich data material with detailed descriptions of the observation setting, including, but not limited to, a focus on the coordination of development activities (see Table 1). The protocol template is included in Appendix A.

Interviews. In addition to the field observations, we conducted 37 interviews to gain a deeper understanding of the case. Some interviews were conducted on the same day. Interview days are illustrated by the long bars in Figure 2. Interviews were held as open conversations in a semi-structured format. We used the same interview guide throughout the data collection (see Appendix B). The interview guide was slightly modified to focus on the disciplinary area of each role (for example, product owners were asked more about clients and products, whereas architects were asked more about technical architecture). Concerning our focus on change and coordination, the questions remained the same.

Example questions include: “*What challenges do you see now and in the future in the development program?*”, “*How has your role changed over time?*” and “*What do you think have been the biggest developments here in relation to coordination across teams?*” We interviewed 25 individuals in total. Six participants, four program managers, and two architects, were interviewed twice. One person, the agile methods specialist, was interviewed six times. These follow-up interviews, held approximately bi-monthly, were more conversational and did not follow the same interview guide as the other interviews. We included these interviews as they contributed to understanding change and coordination over time in the program. The interviews lasted 50 minutes on average. The first author conducted 29 interviews, the second author conducted two interviews, and the six interviews with the agile method specialist were group interviews conducted by the first, second, and third authors. The first author translated interview quotes from Norwegian to English, and all authors checked the quality of the translation.

Supplementary material. During the fieldwork, we were given access to Entur’s internal digital communication tool, Slack, and much of the development process documentation through JIRA and Confluence. The supplemental material allowed us to follow the use of these coordination mechanisms in real-time and record particularly interesting written conversations or documentation pages and provided the ability to search in past conversations and records. For example, we were able to follow in detail the introduction of the new Slack guidelines described in Section 4.3, not only through interviews and observations but also to see the actual implementations in the Slack channels as they took place. In addition to these major sources of information, we collected company presentations, and meeting invitations and agendas sent to us via e-mail.

3.3 Data Analysis

We used thematic analysis (Braun and Clarke 2006, 2012) to analyze the data. We chose this analytical approach because it is a flexible method that allows the researcher to handle a large data set like ours. Thematic analysis is also well-suited for interpretive research because it recognizes the active role of the researcher in shaping the analysis and the findings (Braun and Clarke 2012). Thematic analysis allows researchers to work systematically to identify and analyze commonalities across large and varied project data. The method is flexible because the six phases are iteratively conducted. This means the data can be analyzed while new material is added, which was suitable for our longitudinal fieldwork. Typically conducted following a six-phased, iterative process, the method allows for a deep analysis where results are grounded

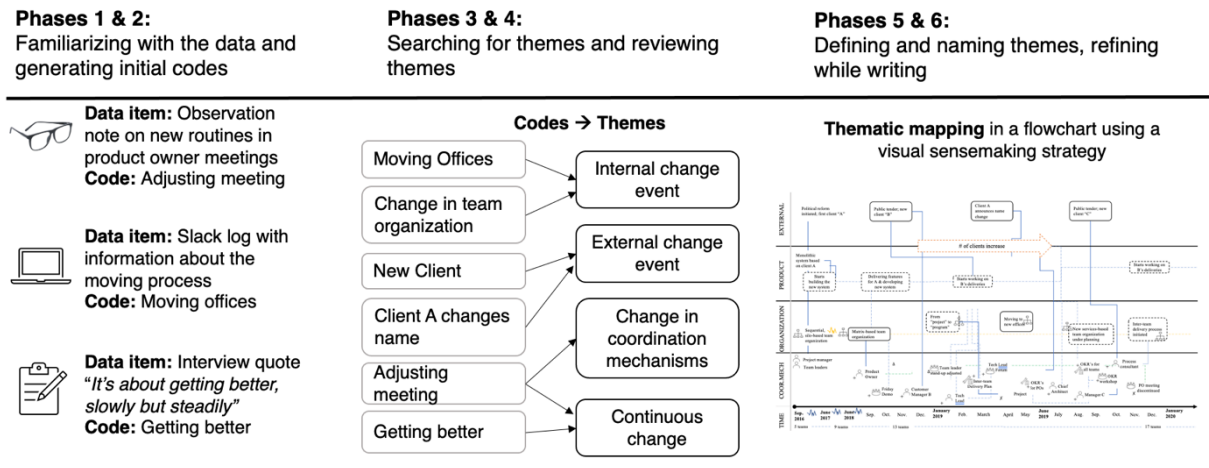


Figure 3. Illustration of the Thematic Analysis.

in the data (Braun and Clarke 2012). Within software engineering, the method is widely used to provide a deeper understanding of the content and meaning of data (e.g., Wohlin and Aurum 2015; Munir et al. 2016; Berntzen et al. 2022; Hussain et al. 2022; Ågren et al. 2022). The following sections provide more information about how we conducted the analysis. Figure 3 illustrates the thematic analysis process.

A thematic analysis is ideally both inductively and deductively guided, thereby ensuring strong links to the data material and the existing literature on the subject. In practice, the analysis is often more strongly guided by one of the approaches (Braun and Clarke 2006, 2012). In our case, while we ensured covering both approaches across the phases of the analysis by using Jarzabkowski et al. (2012)’s model of how coordination mechanisms are created from disruptive events, it was our data that most strongly guided the analysis and the resulting themes in that we did not limit our analysis to only look for themes related to this model. Instead, we kept an open mind as to what other change aspects were present in the data. This made us look beyond change events only to notice how continuous change over time influenced the coordination mechanisms in the case.

Phases 1 and 2: (re-)familiarizing with the data and generating initial codes. During these phases, we reviewed the complete data material from the perspective of change, transition, and evolution of the program and its coordination mechanisms over time. Because the data had been used previously for other studies, we already had a collection of existing coordination mechanisms used in the case. The first author re-read the interview transcripts, field notes, meeting observations, and supplementary material and made analytical notes along the way. These were shared and discussed with the rest of the author team. Specifically, the first three authors discussed the opportunity to describe change in coordination over time during the fieldwork. Following this, the first and fourth authors coined the idea to describe and analyse how coordination mechanisms evolved over time during in-person analysis workshops. Following these initial activities, we proceeded to generating initial codes. As illustrated in the left-hand side of Figure 3, at this stage, we used broad, descriptive codes. For example, a Slack log containing the discussion of the office moving was assigned the code ‘moving offices’. The initial coding phases ended when we had assigned relevant codes to all data items.

Phases 3 and 4: Searching for themes and reviewing themes. During the third phase, we shifted from generating codes to searching for themes. Themes are defined as prevalent patterns within the data, that is, recurring instances of similar types (Braun and Clarke 2006), for example, types of changes that happen outside the organization. We searched for such patterns by grouping and re-grouping the codes from the first and second phases. We identified several lower-level change themes, including changes related to public tenders and clients, changes in the organization of the product, changes in the internal team structures, changes in meeting practices, roles, and tools and artifacts (i.e., coordination mechanisms), changes in the physical location, and much more. Shifting to the fourth phase, we examined the change themes in detail and combined themes that could be grouped under larger themes in light of the data and the definitions of change presented in Section 2 (Jarzabkowski et al. 2012; Langley et al. 2013). We ended up with three high-level change themes, namely external and internal *change events* and *continuous change* (see Table 2), as well as *changes in coordination mechanisms* (see Table 3). Here, we used the categories from the taxonomy of inter-team coordination mechanisms (Berntzen et al. 2022), that is, *meetings*, *roles*, and *tools and artifacts*, but focused this analysis on changes in the mechanisms. The middle part of Figure 3 provides a simple representation of how we arranged codes into themes.

Phases 5 and 6: Defining and naming themes and producing the report. The final two phases of a thematic analysis tend to intertwine as findings are often put to scrutiny through writing up the final report (Braun and Clarke 2012), which was also the case in our analysis. We selected interview quotes, field note passages, and supplemental material for presentation and related the findings back to the research question. Themes were refined during the writing process as all authors wrote, read, and discussed the material. As illustrated by the right-hand side of Figure 3, we organized the results using a visual mapping strategy (Langley, 1999) (see Figure 4), which helped us further refine the presentation of our findings. At this point in the analysis, we conducted member checks by providing Entur representatives with the draft to receive their input to ensure that the findings also held practical relevance.

4 Findings

In presenting our findings, we first describe change events identified from the longitudinal data. Next, we provide detailed examples of changes in the coordination mechanisms used in the program over 1.5 years. We explain how each example relates to the *disruption of existing ways of coordinating*, *orienting to absences in coordinating* and *making new efforts to coordinate, which creates new patterns of coordinating* (Jarzabkowski et al. 2012). Tables 2 and 3 summarizes the change events and changes in coordination mechanisms. Figure 4 presents a process flowchart consisting of five lanes. The first lane represents the organization's external environment, and the second and third lanes represent the product and the internal organizational environment. To illustrate the organization's growth, we have included a representation of the number of teams between the second and third lanes. The fourth lane shows the coordination mechanisms used at Entur that changed over time in relation to changes identified from the analysis and showcased in the above lanes. Finally, the fifth lane, 'time,'

displays the timeline for the changes. In Figure 4, each change event is indicated by a circle containing a letter and a number (e.g., E1, I2) corresponding to the sub-sections in Sections 4.1 and 4.2. In addition, the arrows with dotted lines that run alongside the lanes represent continuous or ongoing changes in the product or the organization, described in Section 4.3. Arrows are drawn from each box to the relevant mechanisms to symbolize the relationship between a change event and a coordination mechanism. In a large-scale development program like Entur, there are more changes than can be described in a report or presented in a flowchart. We, therefore, selected the most compelling examples related to our research questions.

4.1 External Change Events

The first theme is related to changes that took place in Entur's external environment. We consider them 'events' because they happened at a specific point in time. These are presented in the upper lane of Figure 4. One of the product owners explained the importance of context surrounding the organization: "Our whole external context, with the public reform and all it entails, moving from one software system to another, it has all been decided by external circumstances. Our maneuverability is shaped by it" [I01, Product Owner]. In the following, we present three notable external change events.

External change event 1 (E1): New client. When Entur was established in 2016, "Client A" was the only railway operator in Norway. Therefore, much of the development of the new software system during Entur's early development phases were based on Client A's needs and prioritizations. The situation changed in October 2018 when an international railway operator, "Client B," won a public tender following the public transportation reform. Client B would now establish in Norway and use Entur's sales system. Following the announcement was a period of preparation before Entur started working with Client B's requirements in early 2019. These requirements were added on top of other priorities. "*There will certainly be tough deadlines towards Client B, too! Not preparing for that would be naïve*" [I03, Program Manager].

Adding a new client disrupted existing ways of coordinating in that more dependencies was added, and the need for overview across the teams increased. In relation to the onboarding of Client B, new coordination mechanisms were introduced as a response to orienting to the new coordination needs, and new patterns of coordinating were formed by the introduction of new coordination mechanisms. These included a new customer manager role to complement the customer manager of Client A and two artifacts, an inter-team backlog, and an inter-team delivery plan to track which teams worked on what deliveries. The new role and artifacts can be seen in the 'Coordination mechanisms' lane at the bottom of Figure 4, following the lines from the upper lane.

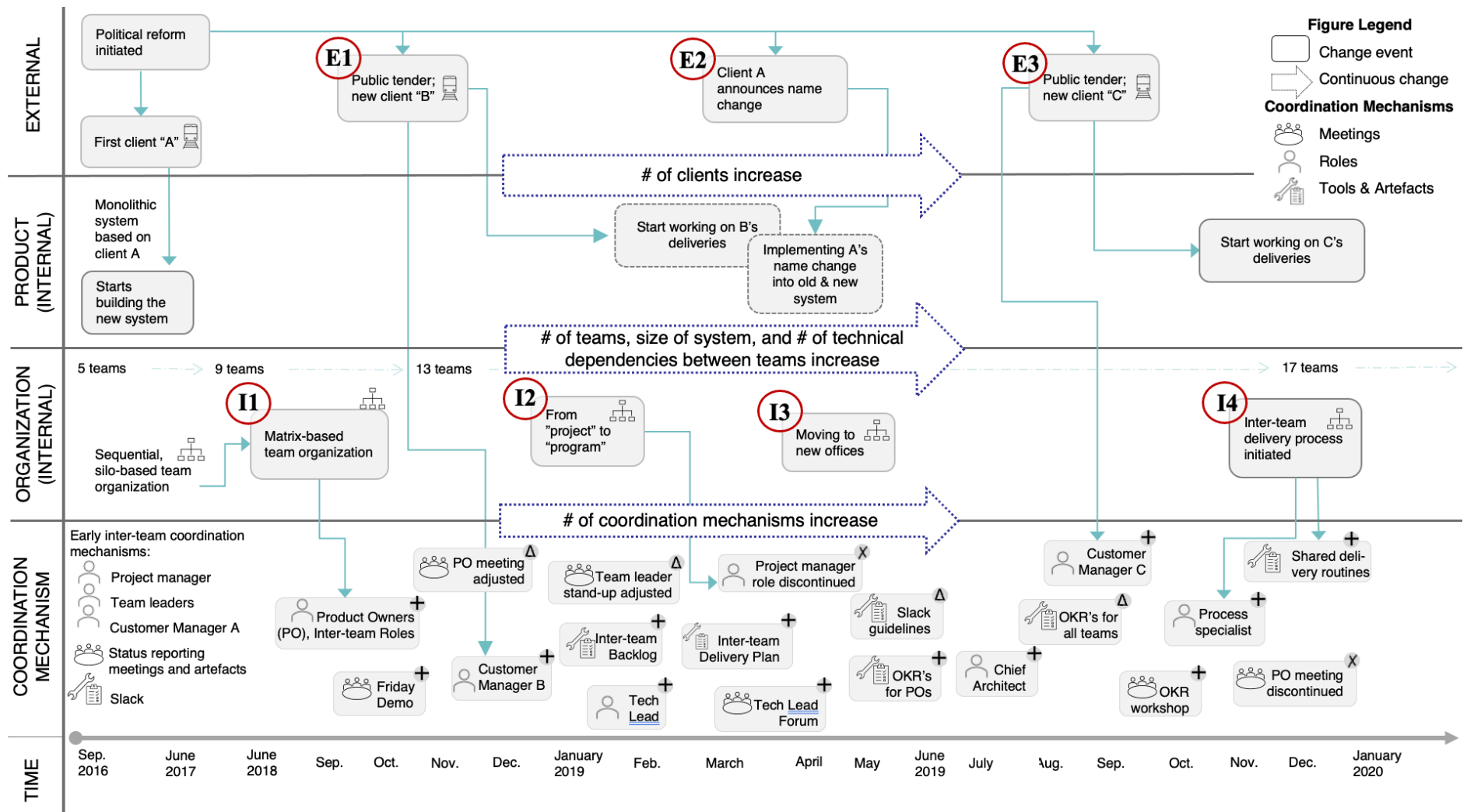


Figure 4. Change events and changes in coordination mechanisms represented through a visual mapping strategy (Langley, 1999). The red circles containing letters and numbers (e.g., E1, I2, etc.) correspond to the sub-sections in sections 4.1 and 4.2. New coordination mechanisms are indicated with a plus sign (+), discontinued mechanisms with a cross (X), while changes to existing mechanisms are indicated with a delta (Δ)

External change event 2 (E2): Client name change and rebranding. Another important change event occurred in March 2019, when Client A announced that they intended to rebrand, changing their name, logo, and visual appearance. This strategic rebranding was a huge change event for Client A, who had had their previous name for almost two centuries. It also directly impacted Entur, who had to adapt both the old and the new systems, as the old name was hardcoded in the legacy code throughout the old system. During a team leader stand-up meeting in late March 2019, the agile method specialist informed the team leaders of the change: *“In a month, Client A’s web pages will close, and a new web page with the new brand will launch. To us, this means that everything that is visible externally needs to be renamed and visually appear as Client A’s new name. [...] the teams need to implement changes in the code. For example, the names of all product IDs in the system must be updated.”* [Meeting observation, March 2019]. The event is presented in the middle of the ‘external’ and ‘product’ lanes of Figure 4. This change event illustrates how an uncontrollable external environment had implications for the teams and the system development. While no new coordination mechanisms were added, we observed how existing mechanisms were updated to accommodate the change in coordination needs and the extra work associated with the name change. For instance, we observed that new lanes were added to delivery plans and roadmaps (see Figure 6 for an example of a physical roadmap) and that the name change was discussed regularly at inter-team stand-up meetings.

External change event 3 (E3): Another new client. A third change event took place in mid-2019, when “Client C” won another public tender, resulting in a change process in late 2019 similar to that of Client B’s entrance. The onboarding of Client C started in early 2020, about at the time when our data collection period ended. Concerning changes in coordination mechanisms, a new customer manager was added to support Client C. Existing inter-team backlogs and delivery plans were updated to make room for incoming requirements and deadlines from Client C, and in January 2020, a workshop was held looking back at lessons learned from onboarding Client B to further adjust practices in preparation for the third major client.

4.2 Internal Change Events

The second change theme is what we refer to as *internal change events*. As opposed to the external change events, these changes originated within the boundaries of the organization. In Figure 4, these change events can be seen in the ‘Organization’ lane. We present four notable internal change events.

Internal change event 1 (I1): Changes in team organization. When Entur was established in 2016, there were five sequentially organized teams. Each team worked on developing their own part of the system, and there was little to no communication between the teams. *“It was truly bad! But we have worked our way forward little by little. First, we got the priority boards, and then that didn’t work so well. To begin with. But then we started to do something, and things got a little better. Also... the [software] modules were maturing, so we had to start talking*

across teams, and so we have also moved forward in an agile way, sort of" [I13, team leader]. In 2017, a new team matrix-based team organization was established, with nine teams organized according to product delivery areas and inter-team roles [Company presentation, November 2017]. By September 2018, there were thirteen development teams organized under nine delivery areas; examples include Pricing, Sales, Ticketing, and On-board services. This new organization was designed to allow for more and better inter-team coordination in response to the coordination needs following the current size of the program. This also led to the introduction of more coordination mechanisms, as the existing ways of coordinating were no longer efficient. The product owner role was implemented for each of the nine delivery areas, and inter-team roles such as agile method specialist, customer manager, and development manager were formally implemented in the organization matrix. After we concluded our data collection, as Entur continued to scale, there was a need to consider yet another change in the team organization. Rather than being organized according to delivery areas, they would gradually organize according to product areas from 2020 onwards.

Internal change event 2 (I2): From ‘project’ to ‘program.’ Initially, the software development at Entur was organized in a development project referred to as the Leap Project. This project was directly linked to the political reform and primarily focused on building the new software, which was done in parallel with running services on the old sales system inherited from Client A. However, because of the scope and magnitude of work associated with developing a new sales platform while running and maintaining the old system and because new clients were added, the project was expanded into an ongoing development program with no end date rather than running many different projects. However, while the Leap project officially ended by the end of 2018, traces of the project organization remained for some time as the program members oriented to new ways of coordinating, both in terms of coordination mechanisms and way of thinking. A product owner explained: *"We have simplified how and how much the teams report, but all the mechanisms are still there"* [I20, Product Owner]. For

Table 2. The major change themes, based on Jarzabkowski et al. (2012) and Langley et al. (2013).

	Description	Examples
External change event	Time-specific changes taking place in the company's external environment, and the control of which are beyond the organizational boundaries but has implications for actions within the organization.	-Transportation reforms leading to the onboarding of new operators/clients (E1, E3) -Client makes name change and rebrands which impacts the development (E2)
Internal change event	Time-specific changes initiated within the organization, controlled by the organization, and based on pre-planned assessments. Has implications for inter-team and team-level coordination.	-Reorganization of team or organization structure (I1, I2) -Moving offices (I3) -Implementing shared delivery routines (I4)
Continuous changes	These changes have no set dates but occur on an ongoing and ad hoc basis. Can be both internally and externally driven.	-Adjusting meeting practices based on retrospectives (internal) -Picking up "best practices" such as new technology and development methods (external)

example, as shown in Figure 4 and discussed in Section 4.3, the Project Manager role remained until April 2019. This example illustrates that coordination mechanisms are not only created in response to change events and that new patterns of coordinating can include the discontinuation of a mechanism.

Internal change event 3 (I3): Moving offices. In April 2019, a third internal change event occurred when Entur moved offices to a new building. When we began our data collection in August 2018, Entur was located on a single floor in a larger office complex. As they continued to grow, the office space became too small to support the program's need for inter-team coordination. This was reflected in our observations during 2018. Task boards hung wherever they fitted in, and there were few open spaces for informal meetings and socializing. Stand-up and prioritization meetings took place in corridors and were constantly interrupted by people passing. The meeting rooms were too small for any inter-team meeting and too small for many of the development teams (of which the largest counted 16 members). Moreover, due to the lack of space, several of the development teams had to sit off-site, which was an obstacle to efficient inter-team communication and coordination.

The office move was a big change event that took time and effort, but it was necessary for successful unscheduled coordination and communication across teams as the program continued to scale. *“Communication might get better now that all teams are in the same building. Because before, we couldn't walk over to each other, but now we can”* [I13, Team leader]. The new offices spanned two floors, connected by a large open staircase that could be used for informal seating and presentations. They had several large meeting rooms and two large open spaces that allowed for more efficient use of existing coordination mechanisms. For instance, task boards could be displayed in the open space, and inter-team meetings could now be held in well-suited areas (see Figures 5 and 6 in Section 4.3). Despite this upgrade, the new offices were also at the risk of becoming too small as the program continued to scale. *“We just keep growing. We moved to get more space. Now, new desks are constantly being added, and we no longer have a dedicated stand-up room as we need the space for workstations”* [I24, Tech lead].

Internal change event 4 (I4): Inter-team delivery routines. In the large-scale program, teams often worked on the same deliveries or on inter-dependent deliveries. Accordingly, as Entur continued to scale, there was a growing need to align the delivery process across teams. In October 2019, measures were taken to establish an inter-team delivery process with common delivery routines. This was done to improve predictability in deliveries across teams to the clients. However, to keep with an agile way of thinking, these new processes were not implemented overnight but piloted and tested in one central team before scaling further. *“We try it out with a smaller group to see ‘do we see the value of doing this?’ We start off narrow, and if people think it gives value, then it is a good model for testing before we go full scale”* [I19, Program Architect]. We observed the piloting phase conducted in October-December 2019. As part of this phase, several new coordination mechanisms changed. As seen in the lower right corner of Figure 4, new coordination mechanisms included a process specialist role and shared routines for using JIRA and Confluence.

Table 3. Changes in coordination mechanisms. Categories based on Berntzen et al. (2022).

Coordination mechanism category	Description	Examples of changes
Coordination roles	Roles are coordination mechanisms performed by people coordinating with other people that contribute to managing dependencies within or across teams.	-Introducing the product owner role -Discontinuing project manager -Introducing tech lead role -Introducing chief architect role -Adding customer managers
Coordination meetings	Time-boxed or ad hoc arrangements where dependencies are managed by enabling people to discuss, share knowledge and negotiate shared understandings.	-Shortening meetings -Changing meeting scope -Removing meetings -Adjusting meeting focus -Increasing unscheduled meetings
Coordination tools and artifacts	Tools manage dependencies by supporting the development process, while artifacts are by-products of the development process.	-New Slack communication guidelines -Adjusting meeting agendas -Adding Inter-team backlog -Adding inter-team delivery plan

4.3 Continuous Changes in Coordination Mechanisms

In addition to the changes in coordination mechanisms following the change events, there were *continuous changes* in coordination mechanisms that reflected the continuous growth and evolution of the program. Often, these were ongoing changes that went unnoticed. “*You don’t put down in writing that ‘this is how we do things here,’ and then people know what it’s like. It’s more like... it flows a bit. And suddenly, things have changed a little. You just notice, like, ‘oh yes, things have changed’ [laughs]*” [I04, Product Owner]. Table 3 and the bottom lane in Figure 4 illustrate these changes. In the following, we provide examples for each of the coordination mechanisms categories, *roles*, *meetings*, and *tools and artifacts*.

Coordination roles are coordination mechanisms performed by people coordinating with other people that contribute to managing knowledge dependencies within or across teams. Entur had several coordination roles, including team-level roles, such as product owners and team leaders, and inter-team roles, such as customer managers and architects. More coordination roles were added as the program scaled, and some roles changed in response to the program’s growth. As mentioned in Section 4.1, the project manager role was discontinued after Entur changed from ‘project’ to ‘program’. The person who filled this role, an external consultant, left in April 2019. However, it took some time for the developers to adjust to this change. “*The project has long been shut down, and our focus is now on product development. But we notice that the project way of thinking remains, and the idea of the project manager role also remains. After a stand-up last week, a team leader asked: ‘Who’s our project manager now? Who will follow up on us?’*” [I20, Product Owner].

As Entur grew in response to the scaling and development of the new software product, at the same time as the old system was kept in use, the number of technical dependencies increased. This led to an increased need to focus on the software architecture both within and across teams. As a result, the tech lead role was established in all development teams in January 2019. “*The role is about technical coordination and in a way be a person within the team that*

has the knowledge and insight about the team architecture that can discuss and be part of making technical decisions, within the team, and also outside the team” [I21, Tech Lead]. Additionally, a chief architect role was added in June 2019 “responsible for coordinating the architects and be part of deciding the scope of the architecture function at Entur” [I14, Program Architect].

The product owner role was a central role at Entur associated with many changes. As explained in Section 4.2, the role was established during the reorganization in 2017 to correspond with the nine delivery areas. Among the product owners’ primary responsibilities was coordinating priorities towards the overall product deliveries and communicating the needs and prioritizations of each development team at an inter-team level. At first, there was a 1:1 correspondence between the delivery areas and development teams. *“What’s interesting about the product owner role here is that it’s influenced by the situation we’re in. Now and in the future. When we implemented the role, the thought was that the product owners themselves would be part of shaping the role. To own their delivery area and be the CEO of their own product, so to speak” [I06, Program Manager].* However, as the program scaled, this quickly changed such that some product owners became responsible for more than one team.

Importantly, at Entur, the product owners were considered as part of the development teams and not an inter-team role. Even the two product owners who had more than one team each (see Figure 1) were primarily affiliated with the teams rather than with the product owner group. This primary affiliation with the teams represented a challenge for inter-team coordination and prioritization of deliveries across teams. *“They all have the same role. But they perform it very differently. That’s the problem” [I12, Program Manager].* Another manager explained: *“They have no sense of group affiliation. But it’s a point to make coordination across the teams work. And if we say coordination across teams is one of our challenges, that includes the product owners. They don’t seem to talk enough to each other”.* [I03, Program Manager]. During our fieldwork, we witnessed several adjustments of the coordination mechanisms surrounding the product owners in order to manage inter-team dependencies more efficiently. They held quarterly retrospectives where inter-team coordination issues were addressed, and changes and adjustments to improve inter-team coordination was made. Most notably, the prioritization meeting (to be introduced below) but also changes in how they communicated on Slack, what to discuss in their weekly meetings and how they could improve inter-team coordination on an ad hoc basis. All along, there was a promise of change attached to the role. *“I do not believe the product owner role is the same now as next year or the year after. How many product owners do we need today, tomorrow, or in the long run? I think that number will vary [I03, Program Manager].*

Coordination meetings are coordination mechanisms where dependencies are managed by enabling people to discuss, share knowledge, and negotiate shared understandings. At Entur, both scheduled and ad hoc, unscheduled meetings were frequently used to manage dependencies within and across teams. During our data collection, we observed the ongoing adjustment and improvement of coordination meetings. The program members had the autonomy to adjust these mechanisms, which often happened during inter-team retrospectives. For example, the product owner prioritization meeting was adjusted in November 2018 based on input received in a retrospective meeting for the product owners. After this, the product

owners kept experimenting with the meeting format, and in November 2019, they decided to discontinue the meeting. *“The last few weeks, the product owners have had stand-up meetings instead. I asked them if using the [prioritization] task board still made sense, and most said they did not want to use it anymore”* [I03, Program Manager]. Similarly, the team leader stand-up meeting was adjusted in early 2019 following a team leader retrospective. During the retrospective, held in February 2019, some team leaders complained that the stand-ups had become time-consuming reporting meetings. *“In the retro, we decided to only focus on issues relevant across teams, which has saved us some time. So, the stand-up improved, for now at least”* [I13, Team Leader].

Coordination meetings were also added during the study to fit the program’s needs. For instance, an inter-team Program Demo, where teams showcased parts of their development work every week, was established in September 2018. This demo contributed to coordination by enabling shared knowledge across teams. In March 2019, following the implementation of the tech lead role earlier in the year, the first “tech lead forum” was held. This was a bi-weekly meeting for the tech leads and the program architects aimed at sharing knowledge and coordinating technical dependencies across teams. After the forum was established, it took some time to adjust and find the right format. When the forum had been running for some months, a program manager explained: *“In the beginning, not everyone understood their role or wanted to speak up and share their opinion. We wanted to be careful with telling the tech leads what to do, want them to figure it out, and take responsibility themselves. They’re starting to adjust, now we start to see discussions and the type of knowledge sharing that we wanted”* [I12, Program Manager]. The tech lead forum was modeled after the ideal of communities of practice and was planned to establish several such inter-team fora for other inter-team coordination areas, such as software quality and testing and DevOps. *“We wanted to start off with one such forum, not all at once, and see what we more we wanted over time”* [I06, Program Manager].

In addition to these scheduled meetings, unscheduled coordination meetings improved following the office move in April 2019, as there was more open space available and more meeting rooms that enabled spontaneous meetings (see Figure 5). Additionally, the open staircase was used to display inter-team coordination mechanisms (see Figure 6) and enabled easy access to members of other teams. This open staircase was also designed with seating and was used for informal lunches, company presentations, and hangouts.

Coordination tools and artifacts. Coordination tools are coordination mechanisms that manage dependencies by supporting the development process, for example, a chat tool, while coordination artifacts are considered by-products of the development process, for example, documentation. At Entur, coordination tools and artifacts were used widely, both at the team and inter-team levels. Over time, more inter-team tools and artifacts were added to align inter-team coordination. In the past, the teams had their own backlogs and delivery plans, which the product

owners reported on during their prioritization meeting. However, as the program scaled, additional mechanisms were needed. In January 2019, an inter-team backlog was added, and in March, an inter-team delivery plan was put together in response to client growth, as described in Section 4.1. In May/June 2019, Entur started experimenting with OKRs, a goal management

framework that Entur used as a coordination tool. They first tested OKRs with the product owners, and as that gave promising results, it was decided to expand the use of OKRs to involve the team leaders, the architects, and the management group. The goal was that all Entur were to use OKRs by 2020. *“The goal is to gain an overview and to give insights to the organization. And to be able to say, ‘this is where we’re at,’ right. And use this insight to evaluate if something works or not and act [I21, Program Architect].*

In addition to the introduction of new coordination tools and artifacts, existing coordination tools were adjusted as needed. The digital communication tool Slack had been used since the outset in 2016. Slack is built up of channels, which users can create and name within certain boundaries set by the software. Many of the channel names were quite similar. For example, there could be a channel called “ClientA_deliveries” and another called Client_deliveries,” and so forth. By early 2019 the number of channels and various, often similar, names for inter-team and inter-organizational channels became confusing and misleading for Entur employees. This also represented a risk of information being shared with the wrong clients. Accordingly, the need to align communication on Slack resulted in new guidelines for creating and naming Slack channels. During April and May 2019, the new Slack guidelines were introduced. The development teams were encouraged to contribute with input before and during the implementation phase. After the new guidelines were introduced, there was a period of improving the new Slack practices. This example illustrates how the need for more alignment in the coordination process initiated new Slack guidelines, resulting in changes in how the coordination mechanism Slack was used and how the change led to a need to adjust further and improve the use of the coordination mechanism.

As a whole, continuous change and improvement were a part of the program’s core culture and practiced at all organizational levels, from the ‘change workshops’ where managerial-level employees discussed structural and organizational changes to the team-level ‘coffee and architecture’-meetings that some tech leads held to get their team members’ input to the tech lead forum. The latter is an example of employee-driven changes resulting from knowledge sharing of “best practices” across teams. For example, a tech lead told us: *“‘Coffee and architecture’ is a 15-minute meeting I initiated with the team. I picked it up from one of the other tech leads during a tech lead forum. I use it to gain input and involve the team” [I21, Tech Lead].*



Figure 5. The new offices brought new coordination arenas like this multi-purpose



Figure 6. The open office space was used to display coordination mechanisms, such as this delivery plan .

The examples presented in this section demonstrate that at Entur, responding to change was part of everyday work. However, there was also a risk that introducing many initiatives at once could be counter-productive, as employees could perceive that there were too many changes: *“There’s always a lot going on. And that’s a factor: How much do we adjust at the same time? It might actually become stressful to have to get familiar with new things all the time. People might lose interest.”* [I20, Product Owner]. Further, despite the always ongoing changes in coordination mechanisms and Entur’s ability to continuously improve, inter-team coordination remained a challenge in the large-scale program. *“The greatest coordination challenge is synchronization across [teams] in the overall deliveries. If there’s one thing that haunts us, this is it* [I03, Program manager].

5 Discussion

Large-scale agile development projects and programs are often long-term and filled with changes, which has consequences for coordination. Change is often understood either in the form of events or patterns of events (Jarzabkowski et al. 2012) or as a continuous process or flow of activities (Langley et al. 2013). In this study, we took an explorative approach to both views of change. Using the theoretical lens of Jarzabkowski et al. (2012) explaining how coordination mechanisms are created in practice, we sought to better understand change and coordination in large-scale agile by investigating the research question: *What types of organizational changes influence coordination mechanisms in large-scale agile, and how do these mechanisms change over time?*

Through thematic analysis, we identified three themes covering the organizational changes that influence coordination mechanisms, namely external and internal *change events*, and *continuous changes* (Table 2). Further, we illustrated how coordination roles, meetings, and tools and artifacts changed over time (Table 3 and Figure 3). The themes were derived partly based on our conceptual understanding of how coordination mechanisms are formed from disruptive events (Jarzabkowski et al. 2012) and our understanding of the importance of continuous improvement in software engineering (Fitzgerald and Stol 2017). However, through our close and detailed engagement with the data material during the analysis, the findings were strongly linked to the empirical material (Braun and Clarke 2012), and it became clear that our theoretical lens did not cover all aspects of coordination observed in the data. As an outcome, we present a model for understanding change in coordination mechanisms over time in large-scale agile, illustrated in Figure 7. This model extends Jarzabkowski et al. (2012)’s theoretical framework and forms the basis for a model of change coordination mechanisms in large-scale agile. We now discuss the implications of our findings.

5.1 External and Internal Drivers of Change in Coordination Mechanisms

First, our findings show that changes in the external and internal environment lead to changes in the coordination mechanisms used to manage dependencies. Both internal and external change events can be compared with the disruptive events that cause coordination mechanisms to break down. However, Jarzabkowski et al. (2012) focused on one large disruptive event (i.e., the organizational restructuring). We find that events do not need to be of such a magnitude or

disruptive level to lead to a change in coordination mechanisms. Moreover, contrary to Jarzabkowski et al. (2012)'s model, we find that the coordination mechanisms themselves do not necessarily have to 'break down' to change. Often, small adjustments or what we term continuous change were sufficient to cause a substantial change in how dependencies were managed. As illustrated by the example with the product owners' prioritization meeting described in Section 4.3, input received during retrospectives can lead to the adjustment of coordination mechanisms. This focus on continuous adjustment and improvement was a key strength at Entur and a core feature of agile that is not captured by the model of Jarzabkowski et al. (2012).

With respect to the external change events, these were initiated and controlled beyond the organization's boundaries. The upper left corner of Figure 7 shows how external changes drive change in two ways. First, external change events may contribute to the scaling of the development program (upper middle box of Figure 7), thus indirectly contributing to changes in coordination needs as the dependencies increase in number and complexity. The external changes caused by the public transportation reform and the addition of clients due to the public tenders can be seen as major drivers of change. These external change events contributed to the continued scaling and growth of the program and, as such, an increase in different types of dependencies (Strode 2016). From scaling follows a change in coordination needs and, subsequently, a change in coordination mechanisms. For example, technical dependencies increased with the size and complexity of the software, but also with the number of teams developing features. Entur established the tech lead role and the tech lead forum in response to the need to coordinate the continuously growing technical dependencies across teams. Additionally, external change events could directly lead to changes in coordination needs without impacting program growth, as with the name change of Client A (see the arrow that leads directly from the upper-left corner to the box in the middle of Figure 7).

Internal change events, such as reorganizing the team structure, or moving to a new office, were also drivers of change in coordination mechanisms. Internal change events can be

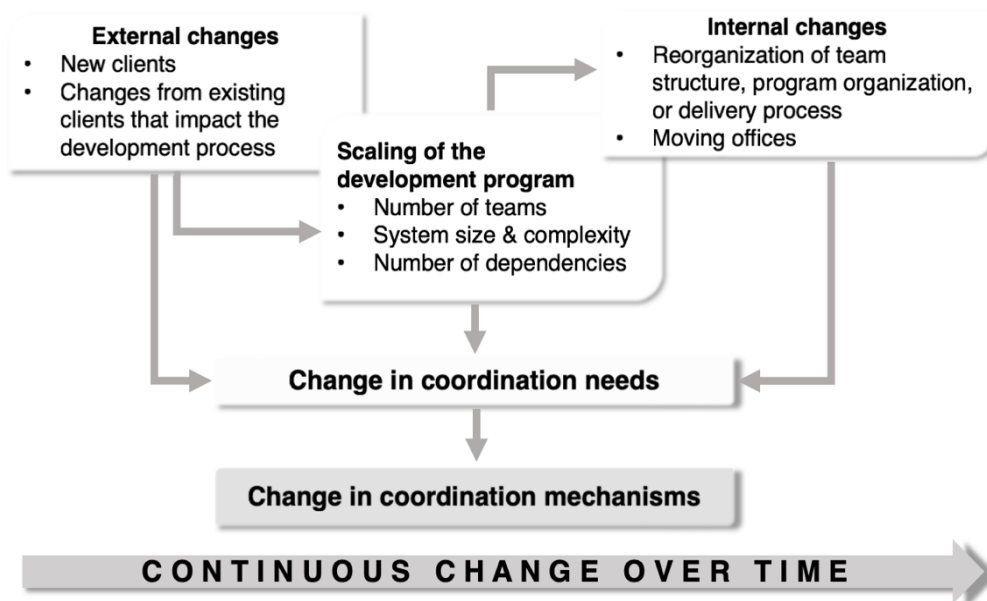


Figure 7. A model of change in coordination mechanisms over time.

understood as ways of adapting to the growth in the client base and the size of the development program. For example, as illustrated by the arrows running from the upper middle to the upper-right corner of Figure 7, the increased number of teams increased the need for inter-team coordination due to more knowledge dependencies across teams. Adding new coordination meetings, such as the Friday Demo, and adjusting existing meetings, such as the team-leader stand-up meeting, were ways of managing inter-team knowledge dependencies. Also, introducing new Slack guidelines was an important way to manage knowledge dependencies when the lack of channel naming conventions caused confusion and the potential risk of information getting out of hand. These findings relate to a study of coordination in global software engineering (Stray and Moe 2020), where the findings showed that the lack of formalized coordination procedures on Slack constituted a challenge to effective dependency management. Additionally, the increase in resource and process dependencies at Entur, caused by more teams needing to coordinate deliveries, was managed by introducing new coordination tools and artifacts, such as inter-team delivery plans and new inter-team coordination roles, such as the process specialist.

We found that changes in coordination mechanisms not only happened reactively because of a disruption or a breakdown in coordination (Cataldo and Herbsleb 2012; Jarzabkowski et al. 2012) but also as a result of Entur's wish to continuously improve their development process (Fitzgerald and Stol 2017). This is in line with Edison et al.'s (2022, p. 14) review, who points out that when organizations seek to improve, "constant change is inevitable" across team structures, processes, tools, and tools metrics. Further, we found that changes were initiated both top-down and bottom-up. In the 'change workshops,' the managers and key inter-team roles discussed organizational and structural issues such as team organization and the establishment of communities of practice (e.g., the tech lead forum), gaining input on "best practice" from research and practice outside the organization. These workshops were examples of top-down drivers of change. From the bottom-up, retrospectives and the tech lead forum were arenas where team representatives could initiate change by discussing and identifying changed coordination needs and how to adapt to them. This is in line with findings from two other large-scale programs in the telecom industry where both top-down and bottom-up approaches to decision-making were used (Moe et al. 2021).

Understanding the dynamics of change in large-scale agile is not easy, as change is not clear-cut. It is difficult to pinpoint, for example, when a "decision" turns into "implementation" and exactly when something changes (Van de Ven and Poole 2005). These issues also pertain to the changes observed in our case study. Although we chose to arrange the themes in terms of change events, there were many instances where it was difficult to see the clear boundaries of the changes. In our results, this is perhaps most clearly illustrated by the change from 'project' to 'program' described in Section 4.1. and Section 4.3. Even though the change was an event that occurred on a specific date, the traces of the project organization remained for some time. In line with the process theoretical perspective (Pettigrew 1990; Langley et al. 2013; Ralph 2018), our findings show that scaling is a constant process that unfolds over time, shaping both organizations and the mechanisms used to manage dependencies. Our study underlines the importance for practitioners in large-scale agile to be mindful of both these aspects and to avoid deciding on a fixed coordination strategy upfront but recognizing the ongoing need to sense and

respond to the situation and continuously improve coordination practices in growing organizations.

5.2 Continuous Growth Requires Continuous Change and Improvement

Second, our findings show that over time, the continuous scaling of the program leads to “more of everything.” This is illustrated by the arrow running across the bottom of Figure 7. This continuous scaling was largely fueled by external events that led to the increase in clients. As the number of clients grows, so does the number of teams needed to develop the system. The more teams, the more dependencies. As such, at Entur, it seemed that change events and the program's growth were closely associated and that both led to a continuous change in coordination needs. In this sense, our findings underscore the importance of dependency awareness in the face of change (Bick et al. 2018), as the ability to sense and respond to changing coordination needs requires understanding how dependencies change.

Our analyses have shown how coordination mechanisms were added, modified, and removed over time in response to changing external and internal environments. The strongest tendency was that the number of coordination mechanisms increased over time. Figure 4 shows that 13 new mechanisms were added, and four mechanisms were adjusted, in relation to the change events described, whereas only two were removed. This finding is interesting in comparison to a recent study where Dingsøyr et al. (2022) identified 27 mechanisms while the case program used a mix of traditional and agile project management techniques. After transitioning to autonomous cross-functional teams, 14 coordination mechanisms were used. In another study, Moe et al. (2018) found that their case started out with many scheduled meetings but that over time, unscheduled meetings were used more. This was explained by the maturing that happened over time. We, on the other hand, found that although some mechanisms were removed, overall, more coordination mechanisms were added over time. We explain this by the continuous growth of the program. Initially, Entur only had one railway client and five teams with low coordination needs, as the number of dependencies between teams was perceived as relatively low. Over time, however, the number of dependencies increased as Entur continued to scale, which required the introduction of more coordination mechanisms.

Our findings can further be related to Fuchs and Hess (2018)'s model, where large-scale agile transformation is understood as episodic phases where each phase is characterized by a radical change followed by a period of incremental changes, and to the lean concepts of *kaikaku* and *kaizen* (radical and incremental change, respectively) (Fitzgerald and Stol 2017). Jarzabkowski et al. (2012) describe phases of destabilization following a change event, during which mechanisms are abandoned, re-formed, or changed before they stabilize. In Entur, however, due to the continuous growth and the unpredictable external environment (i.e., new clients following public tenders and the political backdrop of the transportation reform), such a stabilization period never really seemed to occur. Instead, they needed to rely on sensing and responding to the situation at hand and adapt their use of coordination mechanisms to manage the relevant dependencies at any given time, which often meant adding new mechanisms in response to the continued growth. If the situation stabilizes when the software goes into a maintenance phase and no more clients are added, the need to continue to scale should vanish,

a situation in which they might be able to reduce the number of coordination mechanisms in use.

5.3 Responding to Change by Using the Right Mechanisms at the Right Time

Third, our findings illustrate how having the right coordination mechanisms in place builds resilience to change. As seen in the example of the renaming of Client A (E2), although the event was a significant external change that impacted the coordination needs associated with implementing the name change in the system, Entur was able to handle the event quite smoothly using the mechanisms that were already in place. Information about the event was efficiently distributed to the teams via inter-team coordination mechanisms (e.g., Slack and meetings), and existing inter-team coordination mechanisms, such as task boards and delivery plans, were adjusted as needed to meet the change. This example supports the notion that some coordination mechanisms may be more effective than others in managing certain dependencies (Strode et al. 2012; Stray et al. 2022a).

Our study also provides examples of how to adapt when existing coordination mechanisms do not work effectively (Strode et al. 2012; Strode 2016; Bick et al. 2018). Both the example of adjusting the product owners' prioritization meeting and the need to pay attention to keeping the team-leader stand-up meeting relevant across teams illustrate this. It is already well established in practice that inter-team meetings should focus on sharing relevant information across teams (for example, the Scrum-of-Scrums meeting in the LeSS framework (Larman and Vodde 2016)). Most, if not all, of the meeting participants in Entur's inter-team coordination meetings were knowledgeable and experienced software engineering practitioners who were aware of this. However, in practice, it is difficult to keep this level of discipline and focus and not bring in other information relevant to one's own or one's team's prioritizations. Similar findings have been reported at the team level, where developers have perceived the daily stand-up meeting as too long and not relevant enough (Stray et al. 2016). Using other mechanisms, such as inter-team retrospectives to adjust and adapt regularly, has been found efficient for re-adjusting coordination practices, both at the team level (Strode et al. 2012) and the inter-team level (Edison et al. 2022).

Sometimes the challenging mechanism cannot easily be replaced or modified. In such cases, another way of adapting can be to improve the surrounding coordination mechanisms instead. At Entur, this was most notable with the product owners, where the large and varied group had such different perceptions on how and when to coordinate that it represented an ongoing challenge to the group. This can be explained partly by the team affiliation and partly by the diversity in the group in terms of work background and personalities (Berntzen et al. 2019), but also that the role was given great autonomy in managing their product area. Some studies recommend that inter-team roles, such as product owners, form teams to strengthen inter-team coordination (Bass 2015; Paasivaara et al. 2018). Other studies point to the tension between a strong team focus and a strong inter-team focus in large-scale agile (Gustavsson et al. 2022), where the need for team autonomy must be balanced with the need for inter-team alignment (Dikert et al. 2016; Bick et al. 2018). At Entur, when product owner coordination was challenged, the short-term solution was to adjust the product owner meetings and use of communication tools, and in the long-term, plan to change or even remove the role as a whole.

5.4 Implications for Theory

The findings of this study contribute to calls for more research on how coordination mechanisms emerge, change, and terminate (Jarzabkowski et al. 2012; Moe et al. 2018), as well as expanding research on dependency management at the inter-team and large-scale levels (Strode 2016; Berntzen et al. 2021). This study also raises several arenas for future research, including further development of an emerging theoretical framework for coordination mechanisms in large-scale agile.

Software engineering researchers have been encouraged to adopt a more engaged relationship with theories (Sjøberg et al. 2008; Stol and Fitzgerald 2015). In this study, we adopted a process-theoretical lens, seeking to generate knowledge about *how* changes in coordination mechanisms unfold in a large-scale agile setting (Langley 1999; Ralph 2018). We analyzed our data building on the theoretical framework proposed by Jarzabkowski et al. (2012) that explains how coordination mechanisms are created in practice in response to a disruptive event. We extended this work by including a broader set of change events as our findings show that changes in coordination mechanisms occur not only in response to so-called disruptive events but also in response to internal and external change events, large and small. Moreover, our findings show that not only are coordination mechanisms *created* in response to such changes, but they may also be adjusted or removed altogether.

The model presented in Figure 7 forms the basis for explaining changes in coordination mechanisms in large-scale agile. However, the model needs further theoretical development and empirical investigation (Stol and Fitzgerald 2015), which is an arena for future research. Additionally, future research could build on our findings and do a more thorough mapping of which types of dependencies and coordination mechanisms can be related to which types of change events using existing frameworks (e.g., Strode 2016; Berntzen et al. 2022). Future research can also use insights from this study and our previous work (Berntzen et al. 2022) to study how coordination mechanisms' social, technical, organizational, and physical characteristics change over time in response to changing dependencies. We encourage future work that can contribute to strengthen our findings, for instance by conducting follow-up case studies, or by collecting and analyzing survey data.

5.5 Implications for Practice

Our findings also generate several practical implications that are particularly relevant to large-scale agile programs characterized by high levels of complexity:

- While preparing for all external change events is impossible, having the right coordination mechanisms in place builds resilience to change over time.
- What constitutes an optimal combination of coordination mechanisms will vary over time, as coordination needs are not static.
- When scaling, we recommend using collaboration tools, such as Slack (preferably with communication guidelines), for swift and timely coordination, as face-to-face coordination is not always efficient or even possible.
- Having an overview of the current mix of coordination mechanisms enables companies to sense and respond in a timely and effective manner when coordination needs change.

- Having an explicit and clear focus on continuous improvement of coordination practices (for example, through retrospectives and change-focused workshops) facilitates a flexible way of changing coordination mechanisms in response to change events.

Above all, managers of large-scale agile programs that wish to improve coordination, or manage dependencies effectively in the face of change, should adopt an active and engaged relationship to coordination mechanisms. Agile development welcomes change, recognizing and embracing that it is impossible to avoid change, be it external or internal. At the same time, accurately predicting future coordination needs is nearly impossible. However, it is possible to increase dependency awareness (Bick et al. 2018) through an active and ongoing focus on which coordination mechanisms best address the coordination needs at any given time. This can be achieved by using existing dependency and coordination mechanisms taxonomies (e.g., Strode 2016; Berntzen et al. 2022) to analyze and gain an overview of the coordination situation at hand. Further, awareness of the past, present, and possibly, future changes can be raised by reflecting on how the organization responds to change and how changes have influenced coordination in the past. Table 4 provides practical guidance for conducting such an activity. The questions are based on the analysis for this study and are meant to serve as inspiration for practitioners who wish to deep-dive into understanding change in coordination in their specific organizations.

Table 4. A practical approach to analyzing change in coordination mechanisms

Suggested participants are team representatives and inter-team coordination roles. The analysis can be run in one sitting or in separate steps, depending on the time available and the complexity of the situation.

Step	Goal	Questions
Step 1. Understanding coordination mechanisms	Gaining overview of which coordination mechanisms are currently in use. Ask questions to identify mechanisms	<ul style="list-style-type: none"> • Which meetings do we use to coordinate (between teams)? • Which roles deals primarily with coordination with others? • Which tools and artifacts enable coordination (between teams)? • Which dependencies are managed by these mechanisms? • Are the mechanisms perceived as effective?
Step 2. Understanding past changes	Becoming aware of past change events and continuous changes and how they have influenced coordination. Ask questions to explore and understand.	<ul style="list-style-type: none"> • What changes have we dealt with in the past [insert relevant time period]? Focus on both specific events, as well as changes that have occurred more subtly over time (i.e., continuous changes). • How have these changes influenced how we coordinate? • How long have we used our current coordination mechanisms? When did they appear? Have they changed?
Step 3. Understanding present and future changes	Gaining awareness of ongoing and future changes to potentially be ahead of major changes in coordination needs.	<ul style="list-style-type: none"> • Do we know about any upcoming internal or external change events that will influence our coordination needs? • What changes can we do to existing coordination mechanisms to meet these needs? Will any mechanisms need to be adjusted, removed, or replaced? • Is there a need for other mechanisms? • How will we test any new or adjusted mechanisms and what do we aim to learn?

5.6 Evaluation of Limitations

This study is a qualitative, interpretive case study. We now review limitations of this study by evaluating its credibility, confirmability, and transferability (Guba 1981). These quality criteria are applicable for assessing the trustworthiness of research and are often used within software engineering (e.g., Russo 2021; Hussain et al. 2022).

Credibility. Because this study is interpretive, conducted by humans, and involves human participants, the question of credibility can be raised (Guba 1981; Walsham 2002). We used a range of procedures to make our findings as trustworthy and credible as possible. The ethnographic approach ensured a rich and varied data material (Sharp et al. 2016). The first author conducted the main part of the data collection, which was inevitably subject to her own interpretations and understandings of the case. This is explicitly recognized in interpretive studies; however, it is essential to take measures to safeguard the credibility and trustworthiness of the reporting. To this end, we used an observation protocol and interview guides to sort and systematize our data during collection. We later carefully analyzed the data following established analytical methods (i.e., thematic analysis). Moreover, the three other researchers contributed to the triangulation of the interpretations and the reported findings through ongoing discussions during the fieldwork (authors two and three) and throughout the analytical process (all authors). In longitudinal field studies like ours, the researcher and the participants will, over time, get acquainted with one another, which will influence the research process (Walsham 2002; Crang and Cook 2007). For example, it is impossible to ensure that respondents answer interview questions in an unbiased manner. Here, relying on several data sources and many data points was essential to get as nuanced impressions as possible. Data triangulation was ensured by collecting several data sources. The ongoing and iterative discussions among the authors further contribute to the credibility of our findings. Finally, regular member checks with Entur representatives provide additional trustworthiness (Crang and Cook 2007).

Confirmability. The presented results stem from rich process case data (Langley 1999), where the analysis is based on researcher interpretations. We have gone to lengths preventing that we oversimplified our interpretations of the instances and processes described in this study. By following the six phases of thematic analysis (Braun and Clarke 2006, 2012), we have ensured a rigorous analytical process. However, the interpretive research approach makes it difficult for others to repeat the process to confirm our findings, which is not the goal of such approaches (Walsham 2006). Despite this limitation, it is possible to continue this line of study of change, for example, by using existing dependency (Strode 2016) and coordination mechanism (Berntzen et al. 2022) frameworks to analyze the coordination situation and use a visual mapping strategy as we did to map the developments over time. Future research should aim to reproduce and improve the method and analytical procedures in this study, not to directly replicate the findings but to test the confirmability of the research method.

Transferability. A third limitation relates to the transferability of our findings. This research was conducted within a single organization, and we have focused much on the context-specificity of our case. Other large-scale agile organizations will have a different external and internal context and are, therefore, likely to have a different mix of coordination mechanisms. While we do not claim that our findings are directly transferable to other organizations, the key implications of our findings are likely transferable to other high-complexity large-scale agile

settings. Although the emerging theoretical framework needs further development (Eisenhardt and Graebner 2007; Stol and Fitzgerald 2015), the practical insights from our study provide an empirically based approach to analyzing coordination and change that can aid practitioners in managing dependencies in large-scale agile over time. Even though our case organization did not use any of the large-scale agile frameworks, we believe the findings apply also in companies that have implemented SAFe or any of the other frameworks, because Entur can be considered a ‘critical case’ for coordination in large-scale agile software development. According to Flyvbjerg (2006, p. 230), the generalization characteristic of critical cases can be summed up as “If it is valid for this case, it is valid for all (or many) cases.” In its negative form, the generalization would be, “If it is not valid for this case, then it is not valid for any (or only few) cases.” As such, our findings are theoretically generalizable (Crang and Cook 2007) because other large-scale organizations using agile methods are likely to experience similar coordination challenges and use similar agile practices (Edison et al. 2022).

6 Conclusions

In large-scale software development, change is inevitable because of the complexity and long-term duration of such projects or programs (Edison et al. 2022). Agile practices and the use of coordination mechanisms help navigate the complex dependencies associated with software development at scale. Yet, understanding how change impacts coordination appears important to successful large-scale development (Dingsøyr et al. 2018, 2022). In this study, we addressed the research question, “*What types of organizational changes influence coordination mechanisms in large-scale agile, and how do these mechanisms change over time?*” In previous research, change has been understood either as disruptive events or patterns of events that influence the formation, destabilization discontinuation of coordination mechanisms (Jarzabkowski et al. 2012) or as a continuous process or flow of activities (Langley et al. 2013; Fitzgerald and Stol 2017). In this study, both approaches informed our research question and our analysis.

To investigate changes in coordination mechanisms over time, we analyzed data from 1.5 years of fieldwork using thematic analysis (Braun and Clarke 2006, 2012). We built on the theoretical framework for creating coordination mechanisms in practice proposed by Jarzabkowski et al. (2012) but considered not only disruptive change events and how coordination mechanisms are created, but also how they are adjusted or removed in response to changes in the internal and external organizational environment. Overall, our findings show that external and internal change events were related to changes in coordination needs and, subsequently, changes in coordination mechanisms. Based on our findings, we presented a model of change in coordination mechanisms over time, which we hope will make way for future research on change in coordination over time in large-scale agile. Further, we find that continuous growth requires a constant focus on improvement, which is also related to the continuous change and adjustment of coordination mechanisms. Our findings illustrate that having the right coordination mechanisms in place can contribute to building resilience to change. We suggest that large-scale agile practitioners should actively and continuously focus on coordination mechanisms. This makes it possible to continuously respond to changes in coordination needs, thereby efficiently managing dependencies.

Finally, our research shows that it is possible to change and adapt in response to challenges brought by scaling without relying on a set of mechanisms provided by commercial scaling frameworks. Rather, our research demonstrates that having an organizational mindset of continuous improvement is key to being truly agile in the face of changing external and internal environments in large-scale software development.

Acknowledgments. We wish to extend our thanks to Entur and the informants for opening their workplace to us. This research would not have been possible without their willingness to share their experiences. This research was partly supported by the Research Council of Norway through the Transformit project, Grant Number 321477.

APPENDIX A

Observation protocol. Template based on Crang and Cook (2007).

* Field notes to be taken in a handwritten notebook to be always brought around. Fieldnotes to be written up digitally at the end of each day of fieldwork, or as soon as possible after.

*If any photos taken, either add to this document, or give similar name as this file with a brief description of caption in the document.

Title, date

What is this record about? Meeting, observation of daily work, lunch etc.

Description of activity

– who, what, when, where, why, how? Stick to the facts & descriptions, no analysis-related here.

Direct quotes, snippets of conversation, any textual (SMS, email)

Recognize these are a glimpse of a point in time from a particular perspective.

Reflections

How did fieldwork go today?

Did I influence events in any way?

Did anything go wrong?

Can anything be done differently next time?

How do I feel about it?

Emerging questions/analysis

Any emerging analytical questions?

Any potential lines of inquiry?

Potential theories that might be useful?

Future action

Anything to follow up?

Any particular people to talk to?

Any new information to obtain?

APPENDIX B

Interview guide.

Background/about the role.

1. What is your role and how long have you had the role?
2. Please tell me about your educational background and previous work experiences
3. How would you describe your role. What are the most important tasks?
4. Has your role changed over time? How?

About the development program.

5. Please tell me about the program.
6. Tell me about the use of agile here. Which methods and practices are used? Within teams, across teams?
7. Can you illustrate the team organization and tell me about how you see it? <Draw on a board or blank sheets.>
 - a. What's going on here? Why did it happen?
 - b. Where are the dependencies? (Now, in the past, in the future)
 - c. How do the teams coordinate with each other?
 - d. With stakeholders outside the team?
8. Has there been any changes to the program organization or team organization?
9. What challenges do you see now and in the future in the development program?

About coordination.

10. In your role, who do you coordinate with on a regular basis?
11. Which coordination practices do you use here? Can you list specific coordination arenas and provide some examples?
12. How is coordination between the teams / developers?
13. What do you think are the most important challenges for coordination across teams? Why?
14. What do you think have been the biggest developments here in relation to coordination across teams?
15. Is there anything else you want to tell that I have not asked about, or do you have any questions?

Follow-up questions for recurring interviews.

- a. Since last time, has there been any changes to the program organization or team organization?
- b. What challenges do you see now and in the future in the development program

Declarations:

Funding and/or Conflicts of interests/Competing interests: The authors report no conflicts of interest.

Availability of data and material: The data material is unavailable due to non-disclosure agreements.

Code availability: N/A

Ethics approval: Study reported to the Norwegian Centre for Research Data

References

- Aldave A, Vara JM, Granada D, Marcos E (2019) Leveraging creativity in requirements elicitation within agile software development: a systematic literature review. *Journal of Systems and Software* 110396. <https://doi.org/10.1016/j.jss.2019.110396>
- Allison I, Merali Y (2007) Software process improvement as emergent change: A structural analysis. *Information and software technology* 49:668–681
- Bass JM (2015) How product owner teams scale agile methods to large distributed enterprises. *Empirical Software Engineering* 20:1525–1557
- Batra D, Xia W, VanderMeer DE, Dutta K (2010) Balancing agile and structured development approaches to successfully manage large distributed software projects: A case study from the cruise line industry. *CAIS* 27:21
- Berntzen M, Hoda R, Moe NB, Stray V (2022) A taxonomy of inter-team coordination mechanisms in large-scale agile. *IEEE Transactions on Software Engineering* 49:699–718. <https://doi.org/doi:10.1109/TSE.2022.3160873>
- Berntzen M, Moe NB, Stray V (2019) The Product Owner in Large-Scale Agile: An Empirical Study Through the Lens of Relational Coordination Theory. In: Kruchten P, Fraser S, Coallier F (eds) *Agile Processes in Software Engineering and Extreme Programming*. Springer International Publishing, Cham, pp 121–136
- Berntzen M, Stray V, Moe NB (2021) Coordination Strategies: Managing Inter-team Coordination Challenges in Large-Scale Agile. In: Gregory P, Lassenius C, Wang X, Kruchten P (eds) *Agile Processes in Software Engineering and Extreme Programming*. Springer International Publishing, Cham, pp 140–156
- Bick S, Spohrer K, Hoda R, et al (2018) Coordination challenges in large-scale software development: a case study of planning misalignment in hybrid settings. *IEEE Transactions on Software Engineering* 44:932–950
- Braun V, Clarke V (2006) Using thematic analysis in psychology. *Qualitative research in psychology* 3:77–101
- Braun V, Clarke V (2012) Thematic analysis. In: *APA handbook of research methods in psychology, Vol 2: Research designs: Quantitative, qualitative, neuropsychological, and biological*. American Psychological Association, Washington, DC, US, pp 57–71
- Carroll N, Conboy K, Wang X (2023) From Transformation to Normalisation: An Exploratory Study of a Large-Scale Agile Transformation. *Journal of Information Technology* 02683962231164428. <https://doi.org/10.1177/02683962231164428>
- Castañer X, Oliveira N (2020) Collaboration, coordination, and cooperation among organizations: Establishing the distinctive meanings of these terms through a systematic literature review. *Journal of Management* 46:965–1001
- Cataldo M, Herbsleb JD (2012) Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering* 39:343–360
- Crang M, Cook I (2007) *Doing ethnographies*. Sage
- Dikert K, Paasivaara M, Lassenius C (2016) Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software* 119:87–108
- Dingsøy T, Bjørnson FO, Schrof J, Sporseem T (2022) A longitudinal explanatory case study of coordination in a very large development programme: the impact of transitioning from a first- to a second-generation large-scale agile development method. *Empirical Software Engineering* 28:1. <https://doi.org/10.1007/s10664-022-10230-6>
- Dingsøy T, Fægri TE, Itkonen J (2014) What is large in large-scale? A taxonomy of scale for agile software development. In: *International Conference on Product-Focused Software*

- Process Improvement. Springer, Cham, pp 273–276
- Dingsøyr T, Moe NB, Fægri TE, Seim EA (2017) Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. *Empirical Software Engineering* 1–31
- Dingsøyr T, Moe NB, Seim EA (2018) Coordinating Knowledge Work in Multi-Team Programs: Findings from a Large-Scale Agile Development Program. *Project Management Journal* 49:64–77
- Dingsøyr T, Nerur S, Balijepally V, Moe NB (2012) A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software* 85:1213–1221. <http://dx.doi.org/10.1016/j.jss.2012.02.033>
- Edison H, Wang X, Conboy K (2022) Comparing Methods for Large-Scale Agile Software Development: A Systematic Literature Review. *IEEE Transactions on Software Engineering* 48:2709–2731. <https://doi.org/10.1109/TSE.2021.3069039>
- Eisenhardt KM, Graebner ME (2007) Theory building from cases: Opportunities and challenges. *The Academy of Management Journal* 50:25–32
- Espinosa JA, Slaughter SA, Kraut RE, Herbsleb JD (2007) Team knowledge and coordination in geographically distributed software development. *Journal of management information systems* 24:135–169
- Fitzgerald B, Stol K-J (2017) Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software* 123:176–189
- Flyvbjerg B (2006) Five misunderstandings about case-study research. *Qualitative inquiry* 12:219–245
- Fowler M, Highsmith J (2001) The Agile Manifesto. <http://agilemanifesto.org/>
- Fuchs C, Hess T (2018) Becoming agile in the digital transformation: The process of a large-scale agile transformation. In: *Proceedings of the 39th International Conference on Information Systems (ICIS 2018)*
- Gregor S (2006) The Nature of Theory in Information Systems. *MIS Quarterly* 30:611–642. <https://doi.org/10.2307/25148742>
- Guba EG (1981) Criteria for assessing the trustworthiness of naturalistic inquiries. *Ectj* 29:75–91
- Gustavsson T (2019) Dynamics of Inter-Team Coordination Routines in Large-Scale Agile Software Development. In: *Proceedings of the 27th European Conference on Information Systems (ECIS)*. Uppsala, pp 1–16
- Gustavsson T, Berntzen M, Stray V (2022) Changes to team autonomy in large-scale software development: a multiple case study of Scaled Agile Framework (SAFe) implementations. *International Journal of Information Systems and Project Management* 10:29–46
- Hoda R, Noble J (2017) *Becoming agile: a grounded theory of agile transitions in practice*. IEEE Press, pp 141–151
- Hoda R, Salleh N, Grundy J (2018) The rise and evolution of agile software development. *IEEE software* 35:58–63
- Hussain W, Perera H, Whittle J, et al (2022) Human Values in Software Engineering: Contrasting Case Studies of Practice. *IEEE Transactions on Software Engineering* 48:1818–1833. <https://doi.org/10.1109/TSE.2020.3038802>
- Jarzabkowski PA, Lê JK, Feldman MS (2012) Toward a Theory of Coordinating: Creating Coordinating Mechanisms in Practice. *Organization Science* 23:907–927
- Kalenda M, Hyna P, Rossi B (2018) Scaling agile in large organizations: Practices, challenges, and success factors. *Journal of Software: Evolution and Process* 30:e1954
- Kwan I, Schroter A, Damian D (2011) Does socio-technical congruence have an effect on software build success? a study of coordination in a software project. *IEEE Transactions on Software Engineering* 37:307–324

- Langley A (1999) Strategies for theorizing from process data. *Academy of Management review* 24:691–710
- Langley A, Smallman C, Tsoukas H, Van de Ven AH (2013) Process studies of change in organization and management: Unveiling temporality, activity, and flow. *Academy of management journal* 56:1–13
- Langley A, Truax J (1994) A process study of new technology adoption in smaller manufacturing firms. *Journal of Management Studies* 31:619–652
- Larman C, Vodde B (2016) *Large-scale scrum: More with LeSS*. Addison-Wesley Professional
- Lin B, Zagalsky A, Storey M-A, Serebrenik A (2016) Why Developers Are Slacking Off: Understanding How Software Teams Use Slack. In: *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*. ACM, New York, NY, USA, pp 333–336
- Madampe K, Hoda R, Grundy J (2022) The Emotional Roller Coaster of Responding to Requirements Changes in Software Engineering. *IEEE Transactions on Software Engineering* 1–1. <https://doi.org/10.1109/TSE.2022.3172925>
- Malone TW, Crowston K (1994) The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)* 26:87–119
- March JG, Simon HA (1966) *Organizations*. John Wiley & Sons, New York
- Mintzberg H (1989) *Mintzberg on management: Inside our strange world of organizations*. Simon and Schuster
- Moe NB, Dingsøyr T, Rolland K (2018) To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software development. *International Journal of Information Systems and Project Management* 6:45–59
- Moe NB, Šmite D, Paasivaara M, Lassenius C (2021) Finding the sweet spot for organizational control and team autonomy in large-scale agile software development. *Empirical Software Engineering* 26:101. <https://doi.org/10.1007/s10664-021-09967-3>
- Murray E, Treweek S, Pope C, et al (2010) Normalisation process theory: a framework for developing, evaluating and implementing complex interventions. *BMC medicine* 8:1–11
- Niven PR, Lamorte B (2016) *Objectives and key results: Driving focus, alignment, and engagement with OKRs*. John Wiley & Sons
- Okhuysen GA, Bechky BA (2009) 10 coordination in organizations: An integrative perspective. *Academy of Management annals* 3:463–502
- Paasivaara M, Behm B, Lassenius C, Hallikainen M (2018) Large-scale agile transformation at Ericsson: a case study. *Empirical Software Engineering* 1–47
- Pettigrew AM (1990) Longitudinal field research on change: Theory and practice. *Organization science* 1:267–292
- Ralph P (2018) Toward methodological guidelines for process theories and taxonomies in software engineering. *IEEE Transactions on Software Engineering* 45:712–735
- Runeson P, Höst M (2008) Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14:131. <https://doi.org/10.1007/s10664-008-9102-8>
- Russo D (2021) The Agile Success Model: A Mixed-methods Study of a Large-scale Agile Transformation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30:1–46
- Sharp H, Dittrich Y, C. R. B. de Souza (2016) The Role of Ethnographic Studies in Empirical Software Engineering. *IEEE Transactions on Software Engineering* 42:786–804
- Sjøberg DI, Dybå T, Anda BC, Hannay JE (2008) Building theories in software engineering. In: Shull F, Singer J, Sjøberg DIK (eds) *Guide to advanced empirical software engineering*. Springer, pp 312–336

- Spiegler SV, Heinecke C, Wagner S (2021) An empirical study on changing leadership in agile teams. *Empirical Software Engineering* 26:1–35
- Stol K-J, Fitzgerald B (2015) Theory-oriented software engineering. *Science of Computer Programming* 101:79–98
- Stray V, Moe NB (2020) Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and Slack. *Journal of Systems and Software* 170:110717. <https://doi.org/10.1016/j.jss.2020.110717>
- Stray V, Moe NB, Strode D, Mæhlum E (2022a) Coordination Value in Agile Software Development: A Multiple Case Study of Coordination Mechanisms Managing Dependencies. In: *Proceedings of the 15th International Conference on Cooperative and Human Aspects of Software Engineering*. Association for Computing Machinery, New York, NY, USA, pp 11–20
- Stray V, Moe NB, Vedal H, Berntzen M (2022b) Using Objectives and Key Results (OKRs) and Slack: A Case Study of Coordination in Large-Scale Distributed Agile. <https://doi.org/10.36227/techrxiv.16892161.v1>
- Strode DE (2016) A dependency taxonomy for agile software development projects. *Information Systems Frontiers* 18:23–46
- Strode DE, Huff SL, Hope B, Link S (2012) Coordination in co-located agile software development projects. *Journal of Systems and Software* 85:1222–1238
- Thompson JD (1967) *Organizations in action: Social science bases of administrative theory*. McGraw-Hill
- Uludağ Ö, Philipp P, Putta A, et al (2022) Revealing the state of the art of large-scale agile development research: A systematic mapping study. *Journal of Systems and Software* 111473
- Van de Ven AH, Delbecq AL, Koenig Jr R (1976) Determinants of coordination modes within organizations. *American sociological review* 322–338
- Van de Ven AH, Poole MS (2005) Alternative approaches for studying organizational change. *Organization studies* 26:1377–1404
- Walsham G (2002) Interpretive Case Study in IS Research: Nature and Method. In: Myers MD, Avison D (eds) *Qualitative Research in Information Systems*. Sage Publications.
- Walsham G (2006) Doing interpretive research. *European Journal of Information Systems* 15:320–330. <https://doi.org/10.1057/palgrave.ejis.3000589>
- Wohlin C, Aurum A (2015) Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Software Engineering* 20:1427–1455



Marthe Berntzen is a Ph.D. candidate in Software Engineering at the department of Informatics, University of Oslo. She holds an M.Sc. degree from BI Norwegian Business School and has four years of industry experience. Marthe's Ph.D. research centers around inter-team coordination in large-scale agile software development. Her research focus on agile methods and practices, teamwork, leadership, and coordination in large-scale and distributed settings. She presents her work in journals and conferences within software engineering, information systems and management.



Viktoria Stray is an Associate Professor at the University of Oslo's Department of Informatics. She also holds a senior research position at SINTEF. Her research interests include agile methods, coordination, global software engineering, software testing, and large-scale development. The focus of her research is to improve the productivity of developers and testers and increase the success of software projects. Stray has a Ph.D. in Software Engineering and has worked several years in the industry participating in some of the largest software development projects in Norway.



Nils Brede Moe is a chief scientist at SINTEF. He works with software process improvement, intellectual capital, autonomous teams, and agile and global software development. He has led several nationally funded software engineering research projects covering organizational, sociotechnical, and global/distributed aspects. Moe received a Dr.Philos. in Computer Science from the Norwegian University of Science and Technology and holds an adjunct position at the Blekinge Institute of Technology in Sweden.



Rashina Hoda is an Associate Professor of Software Engineering at Monash University, Melbourne. Rashina specializes in human-centered software engineering, including agile transformations, self-organizing teams, agile project management, and large-scale agile, and has introduced Socio-Technical Grounded Theory (STGT) as a modern variant of traditional Grounded Theory to software engineering. She serves as an Associate Editor of the IEEE Transactions on Software Engineering and as co-chair for ICSE-SEIS 2023, and previously, on the advisory board of IEEE Software and as PC co-Chair for CHASE2021. For more: www.rashina.com.