# A Taxonomy of Inter-Team Coordination Mechanisms in Large-Scale Agile

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

**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?*

---

- *Marthe Berntzen is with the Department of Informatics, The University of Oslo, Gaustadalléen 23B, 0373 Oslo, Norway. E-mail: Marthenb@ifi.uio.no*
- *Rashina Hoda is with the Faculty of Information Technology, Monash University, Wellington Road, Melbourne, Australia, VIC 3800. E-mail: rashina.hoda@monash.edu*
- *Nils Brede Moe is with SINTEF Digital, Strindveien 4, 7645 Trondheim, Norway. E-mail: Nils.B.Moe@sintef.no*
- *Viktoria Stray is with the Department of Informatics, The University of Oslo, Gaustadalléen 23B, 0373 Oslo, and with SINTEF Digital, Strindveien 4, 7645 Trondheim, Norway. E-mail: stray@ifi.uio.no*

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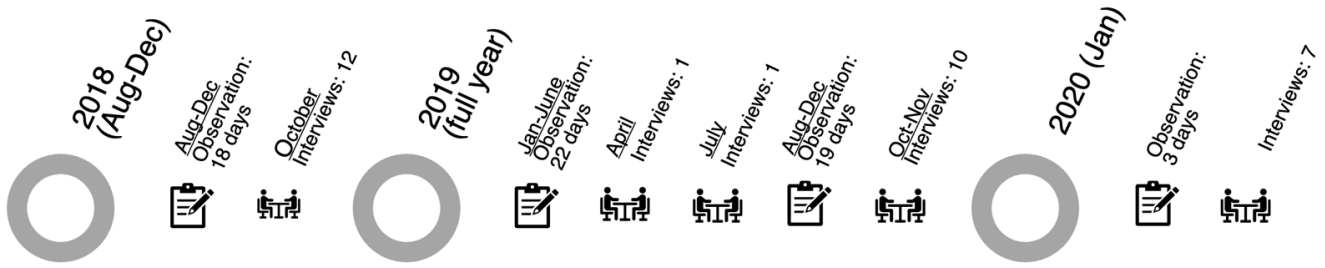
Fig. 1. Overview of the data collection from August 2018 to January 2020

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 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,

### TABLE 1
### DATA COLLECTION DETAILS

| MEETING OBSERVATIONS | 113 hours of observation across 62 days on-site, including: <br> • 10 prioritization meetings <br> • 7 tech lead forums <br> • 7 program demos <br> • 6 product owner meetings <br> • 6 inter-team stand-up meetings <br> • 4 inter-team retrospectives <br> • 2 OKR workshops <br> • 26 ad hoc inter-team meetings <br> • 26 intra-team meetings |
|---|---|
| INTERVIEWS | 31 interviews with 25 participants (mean length 51 minutes). <br> Participants included: <br> • 10 product owners (6 male, mean IT tenure 11.5 years, mean company tenure 1.8 years) <br> • 5 program managers (4 male, mean IT tenure 18 years, mean company tenure 1.6 years) <br> • 4 program architects (4 male, mean IT tenure 19 years, mean company tenure 1.4 years) <br> • 4 tech leads (3 male, mean IT tenure 7 years, mean company tenure 2.4 years) <br> • 2 team leaders (2 male mean IT tenure 9 years, mean company tenure 1.5 years) |
| DOCUMEN-TATION | Slack logs, Confluence documentation, e-mails, internal and external company documents (e.g., presentations, reports) |

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

---

📝 **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

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 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.

---

🗣 **Interview transcript**

**Interview with:** [Participant I15]
**Date:** October 2019
**Place:** [Entur site, small meeting room with whiteboard and whiteboard pens.]
**Interviewer:** Please, tell me about your role in the program?
**I15:** [excerpt only] "I'm here to work with the deliveries in Entur across teams and make them more coherent."
**Code1**: Inter-team role
**Code2:** Deliveries across teams

Fig. 3. Interview transcript extract

**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 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.

---

🔷 **Slack log**

**Date**: April 2019
**Channel**: Open-general
**Topic**: Discussion of new guidelines for channel names

**Person1 [9:31 AM]:** Great work on the new channel names. We still miss input from some teams but we're getting there :slightly_smiling_face:
**Person2 [9:34 AM]:** What about internal channels, how do we name those?
**Person3 [9:53 AM]:** You can use <teamname><app> [i.e., name template for private channel], if you don't mind
**Person 2 [9:43 AM]:** Maybe I'd rather use <openname> [i.e., name template for open channel]... The channel is not just for the team...
**Person 3 [9:45 AM]:** ok, team or open, you know which best yourself :slightly_smiling_face:
**Code1**: Inter-team coordination
**Code2**: Digital communication tool

Fig. 4. Supplemental documentation extract: Slack log

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 1st, 3rd and 4th 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 *inter-team coordination mechanisms* were identified. The 1st, 3rd and 4th 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 *categories* of inter-team coordination mechanisms as well as to the *underlying characteristics* of the coordination mechanisms. The 1st and 2nd 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 1st and 2nd authors were involved in this phase. |
| 5. Defining and naming themes | The specifics of each theme where refined and checked for coherence. Definitions and names were generated for each theme (see Table 3 and Figure 5). The 1st and 2nd 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. |

## 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?" 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

## 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. |
| ⬡ | ORGANI-ZATIONAL | 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. |

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. Through ongoing and iterative discussions, the first two authors examined each coordination mechanisms 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 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.
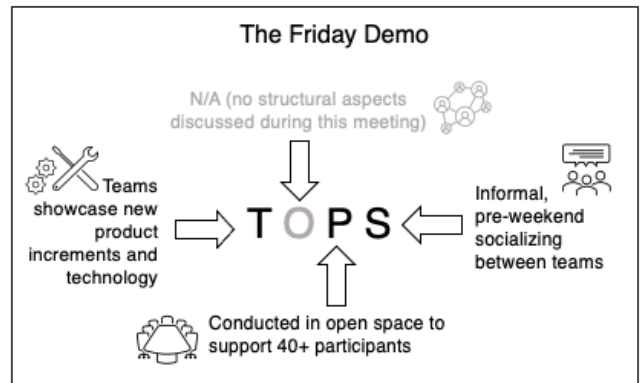


Fig. 5. The technical-organisational-physical-social (TOPS) framework, illustrated by the Friday Demo

## 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.

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

## 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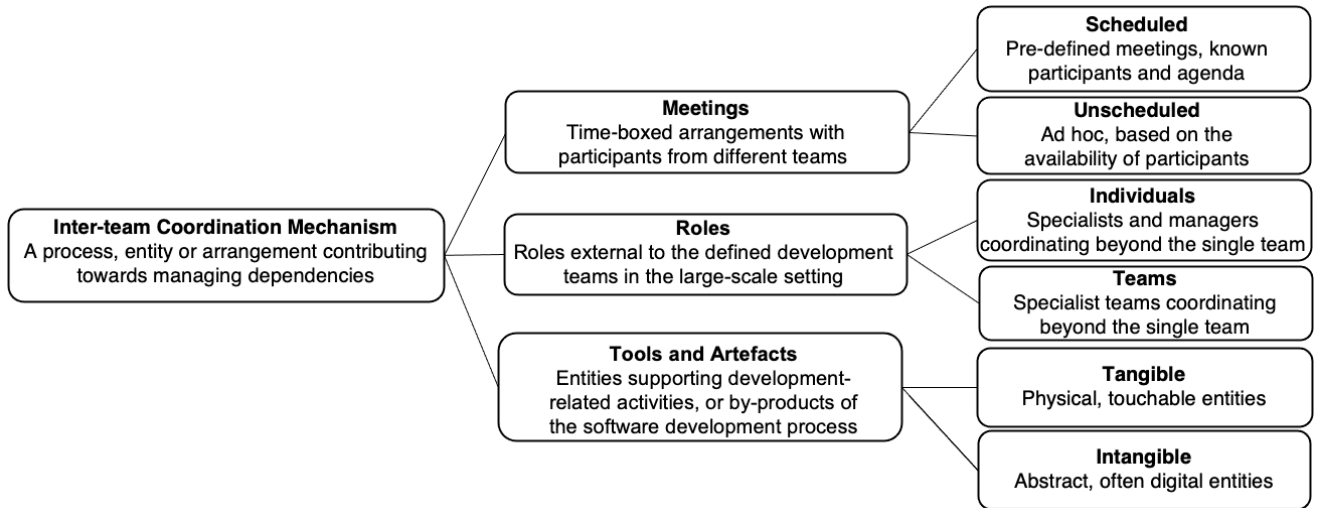 or to the development program as a whole. Inter-team meetings held at Entur included regularly scheduled meetings such as the team leader stand-ups, prioritization meetings, product owner meetings, program architect meetings, Friday demos, and the tech lead forum, which was the regular meeting of the tech lead community of practice. There were also retrospectives for team leaders, product owners, and tech leads, respectively (presented collectively in Table 4); 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 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

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* | ☑ | ✓ | ✓ | ✓ | 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 | ✓ | | ✓ | ☑ | 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* | ✓ | ✓ | ✓ | ☑ | 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 | ☑ | ✓ | ✓ | ✓ | 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 | ☑ | | ✓ | ✓ | 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 | ✓ | ✓ | ✓ | ☑ | 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 | ☑ | ✓ | ✓ | ✓ | 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 | ☑ | ✓ | ✓ | ✓ | 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 | ☑ | | ✓ | ✓ | 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 | ☑ | | ✓ | ✓ | 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 | ☑ | | ✓ | ✓ | 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 | ✓ | ☑ | | ✓ | 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 | ☑ | ✓ | | ✓ | Responsible for agile methods and has overview of requirements, tasks, and prioritizations across teams, thus managing process and technical resource dependencies. |
| Platform team | ☑ | | ✓ | ✓ | 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 | ✓ | ☑ | | ✓ | 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 | ☑ | ✓ | | ✓ | 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 | ✓ | ☑ | | ✓ | 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 | ☑ | | ✓ | ✓ | 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 | ☑ | | ✓ | ✓ | 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* | ☑ | | | ✓ | Tools such as e-mail and Slack, enabling digital communication and information sharing across teams, thus managing resource (technical) and knowledge dependencies. |
| Documentation tools* | ☑ | | | ✓ | 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] | ☑ | ✓ | ✓ | ✓ | 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] | ☑ | | | ✓ | 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] | ☑ | | | ✓ | 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] | ☑ | | | ✓ | Enables communication and information sharing related to overall product delivery milestones across teams, thus managing resource, process, and knowledge dependencies. |
| Roadmap (physical)[a] | ☑ | | ✓ | ✓ | 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] | ☑ | | ✓ | ✓ | 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 [13], are described in section 2.3.

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 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).



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

## 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 Dingsøyr 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 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

### 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 [54] | | ✓ | |
| Bass & Haxby, 2019 [68] | | ✓ | |
| Dingsøyr et al., 2017 [5] | ✓ | | ✓ |
| Dingsøyr et al., 2018 [48] | ✓ | | ✓ |
| Kettunen & Laanti, 2017 [51] | | ✓ | |
| Moe et al., 2018 [15] | ✓ | | |
| Paasivaara et al., 2018 [3] | ✓ | | |
| Paasivaara & Lassenius, 2019 [21] | ✓ | | |
| Sablis et al., 2020 [47] | ✓ | ✓ | ✓ |
| Shastri et al., 2021 [53] | | ✓ | |
| Smite et al., 2019 [23] | ✓ | ✓ | ✓ |
| Stray & Moe, 2020 [20] | ✓ | | ✓ |
| **Our study** | ✓ | ✓ | ✓ |

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.

Dingsøyr 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.
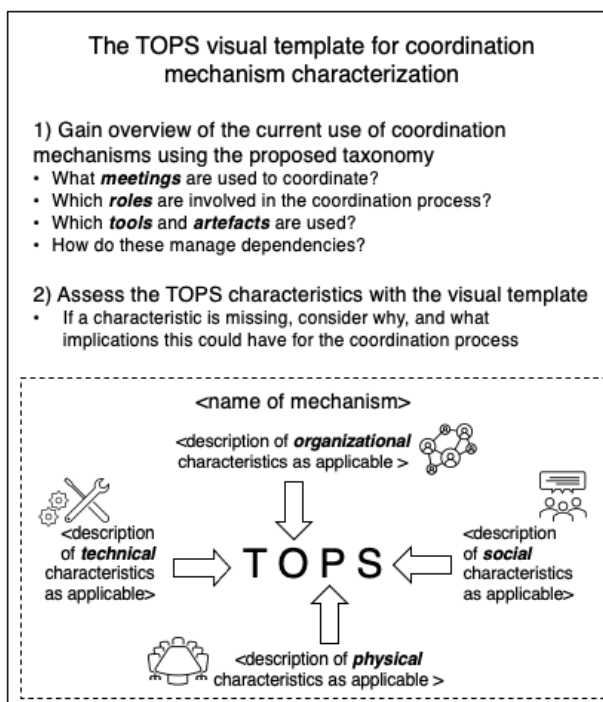


Fig. 8. The TOPS visual template

### 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 engaged in ongoing discussions of insights gained during field work and involved the second author in the data analysis and the taxonomy and framework development. Finally, we conducted member checks with Entur representatives, providing 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

## 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. Dingsøyr, D. Falessi, and K. Power, "Agile development at scale: the next frontier," IEEE Software, vol. 36, no. 2, pp. 30–38, 2019.

[5] T. Dingsøyr, 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. Dingsøyr, 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. Dingsøyr, 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. Dingsøyr, 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. Dingsøyr, 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,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2022.3160873, IEEE Transactions on Software Engineering

BERNTZEN ET AL., THE TOPS FRAMEWORK

21

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 Digital. She has a Ph.D. in Software Engineering from the University of Oslo and an M.Sc. degree in Computer Science from the Norwegian University of Science and Technology. Her research focuses on agile methods, autonomous teams, coordination, global software engineering, innovation, large-scale development. She collaborates with large companies and provides research results on current industry-related topics. She has previously worked for Accenture in Norway's largest software development projects.