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

Marthe Berntzen[1(✉)], Viktoria Stray[1,2], and Nils Brede Moe[2]

[1] University of Oslo, Gaustadalléen 23B, 0373 Oslo, Norway
{marthenb,stray}@ifi.uio
[2] SINTEF, Strindveien 4, 7465 Trondheim, Norway
{viktoria.stray,nils.b.moe}@sintef.no

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

Keywords: Coordination strategies · Coordination mechanisms · Dependency management · Large-scale agile · Inter-team coordination · Software development

1 Introduction

Digital transformation drives new sectors, such as the finance and transportation sectors, to make use of agile development methods, often in large-scale settings. Despite the popularity of agile, there are new and complex challenges associated with agile methods in large-scale settings due to the unavoidable coordination required when many development teams work together [1–3]. When many teams work simultaneously with large code bases, achieving technical consistency across teams, managing stakeholders, balancing a shortage of expert resources, and aligning autonomous teams can become problematic [3, 4]. Practitioners of large-scale agile need to understand how to organize for scale, select optimal large-scale practices, and enable inter-team knowledge sharing [1, 5]. Development teams need to manage dependencies between, for example, requirements, testing, integration, and deliverables, working together with requirement

P. Gregory et al. (Eds.): XP 2021, LNBIP 419, pp. 140–156, 2021.
https://doi.org/10.1007/978-3-030-78098-2_9

engineers, architects, testers, other teams, and support and expert roles, all while keeping in line with the team's goals and prioritizations [3]. Many of these aspects represent coordination challenges, as several parts of the development organization depend on each other to align their efforts to deliver a software product.

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

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

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

2 Background and Related Work

2.1 Managing Dependencies in Large-Scale Agile Development

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

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

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

Table 1. Types of dependencies that can affect agile project progress [24, 25]

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

2.2 Coordination Strategies

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

Xu [13] proposed eight coordination strategies for large agile projects for empirical exploration, focusing on decision-making, communication, and control as relevant dimensions of large-scale coordination and encouraging empirical exploration of these. Li and Maedche [18] conceptually explored coordination strategies within teams in a distributed setting, suggesting that increased communication within the team facilitates shared understandings within the distributed team. Scheerer and colleagues [10] described eight types of inter-team coordination strategies, from purely mechanistic to cognitive and organic, and suggested that future research further explore the concept. These studies recognize that situational factors influence coordination strategies, which should also be relevant to the large-scale inter-team context, where teams are often surrounded by complex organizational contexts [19].

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

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

3 Method and Analysis

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

process and the participants' experiences [22], complemented by in-depth interviews and document analyses.

3.1 Case Description

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

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

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

While the new micro-services-based platform was being developed, PubTrans also delivered services both to their clients (typically public transportation operators) and to the general public through the old, monolithic system. More functionality was added to the new platform continuously and needed to be compatible also with the old system. Many dependencies existed between these systems, and all teams had dependencies to other teams. In addition, there were inter-team knowledge and process dependencies related to, for instance, the delivery sequences. Accordingly, the need for coordination across teams was high.

3.2 Data Collection and Analytical Procedures

During fall 2019, we spent a total of 24 full days at the PubTrans site. The observations consisted of more than 44 h of observation, including a total of 25 meetings. We conducted 12 interviews with team members and program managers. Additionally, we had

Table 2. Data sources

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

frequent informal conversations with the program members. We also inspected documents, logs, and other textual sources for supplemental analysis. The data sources are specified in Table 2. All interviews were tape-recorded based on participants' consent and later transcribed by the first author. The duration was 62 min on average. All interviews were semi-structured, and although the conversations developed naturally, we used an interview guide with questions relating to participants' work habits and inter-team coordination practices. Questions included, *"What challenges do you face working with other teams or roles in the program?," "Can you describe how you interact with members of other teams?," and "What may hinder teams from completing their tasks?"*.

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

The data was coded in NVivo 12 by the first author, who knew the case in detail. To ensure validity, all emerging categories and concepts were negotiated during a series of discussions among the authors, and some of the material was coded by all authors before discussion. The analytical coding proceeded incrementally. During first-cycle coding, we used descriptive and holistic coding to understand "what is going on" in the data [23] and to identify the broad challenges observed and described by the participants. In the second stage, we categorized the challenges that were relevant across teams and identified the various dependencies and coordination mechanisms associated with inter-team challenges using focused coding [23]. Finally, we compared the challenges identified in the first stage with the dependencies and related mechanisms. We considered something a coordination mechanism if it was associated with one or more distinct dependencies, and a coordination strategy when the mechanisms addressed the same set of challenges [9]. As the mechanisms included operated at the inter-team level, we considered them all to be boundary-spanning [9].

3.3 Limitations and Threats to Validity

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

4 Findings

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

4.1 Strategy 1: Aligning Autonomous Teams

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

The missing alignment was also observed when we examined the teams' Jira and Confluence pages; some had well-described processes and documentation, whereas others had little to none. In addition, missing alignment contributed to a lack of technical consistency across teams. *"We have allowed people to develop the new APIs team by team. That means they are not uniform"* [Manager 2]. Although team autonomy was

Table 3. Challenges and coordination mechanisms in the four strategies

	Challenge description	Coordination mechanisms
Strategy 1: Alignment	**Choice of agile methods result in different team routines:** - Different definitions of done - Different development routines - Different testing routines - Lack of technical consistency **Related dependencies:** Process: Activity and Business process dependencies Resource: Technical dependencies	**Synchronization activities**: Inter-team stand-ups and status meetings, tech lead forum **Synchronization tools and artefacts**: Shared routines for deliveries and documentation and testing, common definition of done, test team, platform team to support teams with shared technologies **Structure mechanisms**: Co-location, open office space
Strategy 2: Overview	**Large-scale makes it hard to maintain overview:** - Feeling out of sync with other teams - Problems with information flow - Task-related communication across teams - Locating people and information **Related Dependencies:** Knowledge: Expertise, Task allocation, Requirement dependencies	**Synchronization activities**: Inter-team stand-ups and status meetings, program demo **Synchronization tools and artefacts**: Slack, shared backlog in Jira, organization map on Confluence, program roadmap, Objectives and Key Results **Structure mechanisms**: Open office space, co-location
Strategy 3: Prioritization	**Hard deadlines and many clients lead to prioritization challenges:** - Stakeholder expectation management - Time and delivery pressure - Lack of time to prioritize quality work - Changing prioritizations - Lack of clarity in the prioritization process **Related Dependencies:** Process: Activity dependencies Resource: Entity and technical dependencies	**Synchronization activities**: Inter-team stand-up meetings, Product owner meetings **Synchronization tools and artefacts**: Prioritization task board, shared backlog **Structure mechanisms**: Temporary team arrangements (task force teams, taking on other teams' tasks)
Strategy 4: Architecture	**Complex technical dependencies:** - Two systems in use in parallel - Teams becoming bottlenecks - Large code bases of some teams - Risk of repeating old patterns - Vulnerability for errors **Related Dependencies:** Process: Activity dependencies, Resource: Technical dependencies	**Synchronization activities**: Tech lead forum **Synchronization tools and artefacts**: Objectives and Key Results, platform team **Structure mechanisms**: Temporary team arrangements
	Note. Some coordination mechanisms are recurring across the strategies as they address more than one dependency.	

appreciated, teams also saw the need for alignment across teams: *"It is great that the teams are free and have a lot of responsibility. But it is also essential to have arenas where we can discuss and share knowledge across teams so that it's not spinning out of control"* [Tech lead 2].

The challenges described above were addressed with several coordination mechanisms. PubTrans implemented shared documentation routines on Confluence, and shared delivery routines where a shared definition of done and common testing routines was central. Furthermore, they had established a platform team whose main responsibility was to support the development teams by *"developing functionality across teams, but also handling things like automatic builds, deploying, monitoring and logging overall across the teams"* [Team leader].

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

4.2 Strategy 2: Gaining and Maintaining Overview Across Teams

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

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

Furthermore, Slack channels and direct messages provided the developers with an easy way of sharing knowledge and reaching out to people they did not know. PubTrans also used Objectives and Key Results (OKRs), which is goal-setting framework where objectives and corresponding key results are defined for individual teams and at the organizational level to measure progress over a set time period, typically per quarter [24]. The company used OKRs as a mechanism to provide an overview of the increasingly complex development process. OKRs were formed for all teams during off-site quarterly workshops where product owners, team leaders, and program architects and managers

Fig. 1. The office space with the program's roadmap easily visible

worked iteratively with forming team-specific objectives and key results. Because of the many inter-team dependencies, it was important to compare and discuss OKRs across teams and adjust as needed. *"The goal of using OKRs is to get an overview and gain insight in the organization. OKR allows us to work more structured, and gain overview of 'this is where were we are now'. Then we can assess what to focus on and use it to take action."* [Architect 1]. Together, these activities and artifacts form a coordination strategy for gaining and maintaining overview across teams.

4.3 Managing Prioritization Issues

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

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

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

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

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

4.4 Managing Architecture and Technical Dependencies

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

In developing the new micro-services-based application, it was hard to avoid developing copies of the old system, which, according to a team leader, left them at risk of developing a distributed monolith. *"Overall, we don't have any mechanism to protect us from repeating old patterns. We have some teams that have been able to create something entirely new, but we also have teams that simply re-implement what they have implemented in the past"* [Team leader]. After some time, two teams who developed

key components became bottlenecks. At one point, one team's code base was seemingly large enough to constitute a mini version of the whole platform on which all teams depended. Furthermore, change in one part of the code could have a significant impact on other parts and make the platform vulnerable: *"One risk with developing a distributed monolith with poor error handling is that if one application goes down, the whole system goes down"* [Team leader].

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

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

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

5 Discussion

In this study, we explored the research question: *How are coordination strategies used in large-scale agile to manage inter-team coordination challenges?* We applied the theory of coordination as a guiding lens and extended the coordination strategy concept to the inter-team level. Our findings broaden the application of the theory of coordination beyond single co-located agile teams [9, 20] and answer calls for future research on coordination strategies [10, 13, 18].

According to the theory of coordination, a coordination strategy is a set of agile coordination mechanisms used to manage dependencies [9, 20]. This theoretical lens

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

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

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

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

5.1 Implications for Practice

Our findings generate a number of practical implications. While autonomous teams need to know what others are doing, solve technical dependencies, and align their prioritizations and processes with other teams [28], agile methods offer little specific advice on how this should be implemented in large-scale settings. In line with research on large-scale agile frameworks [16, 29] and hybrid settings [2], we found that coordination needs tailoring to the specific organizational context to cope with uncertainty,

novelty, and complexity [6, 30]. Our results show that the coordination strategy concept is useful for dependency management at scale, and that large-scale agile programs benefit from adapting coordination mechanisms to their specific needs. We suggest that large-scale companies gather insights of their coordination challenges and dependencies across teams and use these to understand their own coordination strategies.

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

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

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

6 Conclusion and Future Research

In this study, we explored the research question of how coordination strategies were used to manage challenges with inter-team coordination in a large-scale agile program with 16 teams. We found the coordination strategy concept useful for studying inter-team coordination in large-scale settings. The concept provides practitioners with an approach that is highly context-specific and flexible and thus suitable for the volatile, complex, and ambiguous large-scale development setting. From our analysis, we found four coordination strategies: 1) aligning autonomous, 2) gaining and maintaining overview across teams, 3) managing prioritization issues and, 4) managing architecture and technical dependencies. We extend the coordination strategy concept to include more practices beyond agile coordination mechanisms, as we found that the mechanisms included in the strategies consisted of both agile practices, such as stand-up meetings and demos, and other practices such as OKRs and a community of practice. Future research could further explore how coordination mechanisms fit together to form coordination strategies, and how to tailor them to contribute to effective coordination in large-scale settings. We also encourage future research to explore coordinator roles in relation to inter-team

coordination strategies. Finally, our on-site access allowed us to explore coordination in a co-located setting. Since then, the workplace has changed, and we encourage empirical research on coordination strategies in distributed settings.

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

References

1. Bass, J.M., Salameh, A.: Agile at scale: a summary of the 8th International Workshop on Large-Scale Agile Development. Presented at the Agile Processes in Software Engineering and Extreme Programming–Workshops (2020)
2. Bick, S., Spohrer, K., Hoda, R., Scheerer, A., Heinzl, A.: Coordination challenges in large-scale software development: a case study of planning misalignment in hybrid settings. IEEE Trans. Softw. Eng. **44**, 932–950 (2018)
3. Sablis, A., Smite, D., Moe, N.: Team-external coordination in large-scale software development projects. J. Softw. Evol. Process. e2297 (2020)
4. Dikert, K., Paasivaara, M., Lassenius, C.: Challenges and success factors for large-scale agile transformations: a systematic literature review. J. Syst. Softw. **119**, 87–108 (2016)
5. Dingsøyr, T., Moe, N.B., Seim, E.A.: Coordinating knowledge work in multi-team programs: findings from a large-scale agile development program. Proj. Manage. J. **49**, 64–77 (2018)
6. Malone, T.W., Crowston, K.: The interdisciplinary study of coordination. ACM Comput. Surv. (CSUR). **26**, 87–119 (1994)
7. Faraj, S., Sproull, L.: Coordinating expertise in software development teams. Manage. Sci. **46**, 1554–1568 (2000)
8. Okhuysen, G.A., Bechky, B.A.: 10 coordination in organizations: an integrative perspective. Acad. Manag. Ann. **3**, 463–502 (2009)
9. Strode, D.E., Huff, S.L., Hope, B., Link, S.: Coordination in co-located agile software development projects. J. Syst. Softw. **85**, 1222–1238 (2012)
10. Scheerer, A., Hildenbrand, T., Kude, T.: Coordination in large-scale agile software development: a multiteam systems perspective. Presented at the 2014 47th Hawaii international conference on system sciences (2014)
11. Smite, D., Moe, N.B., Levinta, G., Floryan, M.: Spotify guilds: how to succeed with knowledge sharing in large-scale agile organizations. IEEE Softw. **36**, 51–57 (2019)
12. Strode, D.E.: A dependency taxonomy for agile software development projects. Inf. Syst. Front. **18**(1), 23–46 (2015). https://doi.org/10.1007/s10796-015-9574-1
13. Xu, P.: Coordination in large agile projects. Rev. Bus. Inf. Syst. (RBIS). **13**, (2009)
14. Uludağ, Ö., Harders, N.-M., Matthes, F.: Documenting recurring concerns and patterns in large-scale agile development. Presented at the Proceedings of the 24th European Conference on Pattern Languages of Programs (2019)
15. Sekitoleko, N., Evbota, F., Knauss, E., Sandberg, A., Chaudron, M., Olsson, H.H.: Technical dependency challenges in large-scale agile software development. In: Presented at the International Conference on Agile Software Development (2014)
16. Gustavsson, T.: Dynamics of inter-team coordination routines in large-scale agile software development. In: Proceedings of the 27th European Conference on Information Systems (ECIS), pp. 1–16, Uppsala (2019)

17. Martini, A., Pareto, L., Bosch, J.: A multiple case study on the inter-group interaction speed in large, embedded software companies employing agile. J. Softw. Evol. Process. **28**, 4–26 (2016)
18. Li, Y., Maedche, A.: Formulating effective coordination strategies in agile global software development teams (2012)
19. Mikalsen, M., Næsje, M., Reime, E.A., Solem, A.: Agile autonomous teams in complex organizations. In: Hoda, R. (ed.) XP 2019. LNBIP, vol. 364, pp. 55–63. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30126-2_7
20. Kanaparan, G., Strode, D.: A theory of coordination: from propositions to hypotheses in agile software development. In: Presented at the Proceedings of the 54th Hawaii International Conference on System Sciences (2021)
21. Yin, R.K.: Case Study Research and Applications: Design and Methods. Sage Publications, Thousand Oaks (2018)
22. Sharp, H., Dittrich, Y., de Souza, C.R.B.: The role of ethnographic studies in empirical software engineering. IEEE Trans. Softw. Eng. **42**, 786–804 (2016)
23. Saldaña, J.: The Coding Manual for Qualitative Researchers. Sage, Thousand Oaks (2012)
24. Niven, P.R., Lamorte, B.: Objectives and Key Results: Driving Focus, Alignment, and Engagement with OKRs. John Wiley & Sons, Hoboken (2016)
25. Berntzen, M., Moe, N.B., Stray, V.: The product owner in large-scale agile: an empirical study through the lens of relational coordination theory. In: Presented at the International Conference on Agile Software Development (2019)
26. Shastri, Y., Hoda, R., Amor, R.: The role of the project manager in agile software development projects. J. Syst. Softw. **173**, 110871 (2021)
27. Moe, N.B., Dingsøyr, T., Rolland, K.: To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software development (2018)
28. Martini, A., Stray, V., Moe, N.B.: Technical-, social-and process debt in large-scale agile: an exploratory case-study. In: Presented at the International Conference on Agile Software Development (2019)
29. Paasivaara, M.: Adopting SAFe to scale agile in a globally distributed organization. In: Presented at the 2017 IEEE 12th International Conference on Global Software Engineering (ICGSE) (2017)
30. Jarzabkowski, P.A., Lê, J.K., Feldman, M.S.: Toward a theory of coordinating: creating coordinating mechanisms in practice. Organ. Sci. **23**, 907–927 (2012)
31. Moe, N.B., Smite, D., Paasivaara, M., Lassenius, C.: Finding the sweet spot for organizational control and team autonomy in large-scale agile software development. Empirical Softw. Eng. (2021)