Empirical Software Engineering (2023) 28:1
https://doi.org/10.1007/s10664-022-10230-6

A longitudinal explanatory case study of coordination in a very large development programme: the impact of transitioning from a first- to a second-generation large-scale agile development method

Torgeir Dingsøyr[1,2] · Finn Olav Bjørnson[1] · Julian Schrof[3] · Tor Sporsem[4]

Accepted: 16 August 2022

Abstract
Large-scale agile development has gained widespread interest in the software industry, but it is a topic with few empirical studies of practice. Development projects at scale introduce a range of new challenges in managing a large number of people and teams, often with high uncertainty about product requirements and technical solutions. The coordination of teams has been identified as one of the main challenges. This study presents a rich longitudinal explanatory case study of a very large software development programme with 10 development teams. We focus on inter-team coordination in two phases: one that applies a first-generation agile development method and another that uses a second-generation one. We identified 27 coordination mechanisms in the first phase, and 14 coordination mechanisms in the second. Based on an analysis of coordination strategies and

Communicated by: Tayana Conte

✉ Torgeir Dingsøyr
torgeir.dingsoyr@ntnu.no

Finn Olav Bjørnson
Finn.o.bjornson@sintef.no

Julian Schrof
julian.schrof@bmw.de

Tor Sporsem
Tor.sporsem@sintef.no

[1] Department of Computer Science, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway

[2] Department of IT Management, SimulaMet, P.O. Box 134, 1325 Lysaker, Norway

[3] Universität der Bundeswehr München, Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany

[4] SINTEF Digital, P.O. Box 4760, Torgarden, NO-7465 Trondheim, Norway

Published online: 08 November 2022

mechanisms, we develop five propositions on how the transition from a first- to a second-generation method impacts coordination. These propositions have implications for theory and practice.

Keywords Large-scale agile development · Coordination mechanisms · Inter-team coordination · Multiteam systems · Software development process · Software engineering

1 Introduction

Coordination is a fundamental challenge in software engineering. Kraut and Streeter (1995, p. 69) stated that '*While there is no single cause of the software crisis, a major contribution is the problem of coordinating activities while developing large software systems'*. In software development, a multitude of dependencies must be managed in a context with high uncertainty about products and technology. Previous studies have focused on coordination in traditional software projects, in global software development and, recently, in agile development.

In the mid-2000s, software engineering research focused on global software engineering, in which coordination amongst distributed teams was a key challenge. The congruence between dependencies and coordination actions is critical both in well-known contexts and in contexts with high uncertainty (Cataldo and Herbsleb 2012). However, an open question concerns what practices are effective. In the paper entitled *Global Software Engineering: The Future of Socio-technical Coordination*, Herbsleb (2007, p. 9) stated that while '*we currently have a number of individual solutions, such as tools, practices, and methods, … we understand as yet very little about the tradeoffs among them, and the conditions of their applicability*'.

In recent years, software engineering research has concentrated mainly on agile software development methods (Dingsøyr et al. 2012; Hoda et al. 2018), in which development is organized as teamwork. Pries-Heje and Pries-Heje (2011) attributed the success of the agile method Scrum to its flexible and efficient coordination structures, its shared list of work tasks in a product backlog and sprint backlog, daily meetings within the team and the use of a visual board to show the status of work. Strode et al. (2012) proposed a coordination model for co-located agile teams, with a focus on synchronization within an agile team, proximity that allows for face-to-face communication and activities targeted at external stakeholders, which she referred to as boundary spanning.

Large IT projects with 10 or more development teams are increasingly using agile methods. Empirical studies show challenges with coordination breakdowns (Bick et al. 2018), lack of awareness and a mismatch between advice in methods and coordination needs over time (Dingsøyr et al. 2018c). Dependencies undermine autonomy, which is essential for agile development teams (Biesialska et al. 2021).

Existing theory is not sufficient to explain coordination in this context, as large-scale agile development has characteristics that differ from those of traditional organizations and distributed development in terms of relying on oral communication, working in teams and frequent changes in coordination mechanisms over time (Dingsøyr et al. 2018a). A systematic literature review on large-scale agile methods reports coordination challenges in large-scale agile development, including synchronizing teams, dealing with communication overload and reducing external distractions (Edison et al. 2021).

Large-scale agile development projects are critical for organizations, representing significant costs and risks. Coordination is critical for project success and on-time delivery (Kula

et al. 2021). The scientific community must provide insight into and advice on coordination in this particular context. Strategies for coordination are described in development methods, and improving our understanding of the effectiveness of these approaches and in which contexts they are effective is essential.

Today, many organizations are changing their approach to large-scale development from what we will define as first-generation large-scale agile methods (Section 2.2.1), which combine practices from project management and agile methods, to more tailored second-generation large-scale agile development methods (Section 2.2.2), which replace practices from project management with practices tailored for managing software development. This change leads to a different approach to coordination, replacing previous solutions, practices and tools. Understanding how the new generation of methods impacts project success is critical. This article focuses on coordination as a significant factor influencing overall project success. More precisely, we examine coordination between teams, which is described in the literature on large-scale agile development as *inter-team coordination* (Edison et al. 2021).

In the following, we present a study of a very large development programme at the Norwegian Labour and Welfare Administration (NAV) with a total cost of about EUR 75 million. The programme, which developed a new solution to automatically process applications for parental benefits, lasted from 2016 until 2019 and had 10 teams working in parallel on development for a long period. We will describe two phases of development, in which a first-generation large-scale agile method was used in the first phase and a second-generation large-scale agile method was applied in the second. We answer the following research question:

> *How is the inter-team coordination strategy impacted by a change from the first- to second-generation large-scale agile development methods?*

This study makes the following three contributions to the literature on coordination in large-scale agile development:

1. Provide a rich empirical description of coordination in a large-scale agile development programme
2. Provide a conceptualisation of methods for large-scale agile development from the first to the second generations
3. Develop a novel theory on the impact of the transition from the first- to second-generation methods on coordination

For the first contribution, a rich description enables readers to make up their own minds on what is relevant in their own situation, provides readers with more background to understand the context of the findings, and also broadens possible use of the study for example in teaching, where students need to build an understanding of industry practice. The second contribution will primarily be helpful for the scientific community in order to distinguish between different types of large-scale development methods studied. For the third contribution, in software engineering, '*we have very few explicit theories [that can] explain why or predict that one method … would be preferable to another under given conditions*' (Johnson et al. 2012). In particular, there are few theories with an empirical basis (Sjøberg et al. 2008); indeed, '*most studies in software engineering pay little or no attention to theory development, and very few studies are based on existing theory*' (Stol

et al. 2016). By developing novel propositions, we provide a contribution towards a theory to understand the impact of large-scale agile methods on coordination.

Section 2 presents the background on large-scale agile development, the definitions of first- and second-generation large-scale agile development methods and an up-to-date literature review of previous relevant work on coordination organized after a model of coordination effectiveness. Section 3 describes the design of the longitudinal explanatory case study, while Section 4 provides a rich description of the programme organization and the findings on coordination for each phase. Section 5 presents coordination in the phases and develops five propositions to answer the research question (shown in Table 10). We also discuss the main limitations. In Section 6, we conclude, show implications for theory and practice and suggest further work.

2 Large-Scale Agile Development and Coordination

Large-scale agile development has drawn significant interest from practitioners (Dingsøyr et al. 2019b) and researchers (Edison et al. 2021; Uludağ et al. 2021), and several new methods, such as the Scaled Agile Framework (SAFe), Large-Scale Scrum (LeSS) and the Spotify model, have been proposed.

We first describe large-scale agile development and first- and second-generation methods. Section 2.2. focuses on coordination—its definition, mechanisms for coordination and a coordination model. We also introduce coordination effectiveness and strategy (choice of coordination mechanisms). Then, we present prior studies on small- and large-scale coordination. Section 2.3 provides research findings on the coordination mechanisms used in large-scale agile development. The presentation is organized after three coordination modes (groups of mechanisms), which are described in Section 2.2.1.

2.1 Large-Scale First- and Second-Generation Agile Methods

Large-scale agile development projects or programmes typically involve many developers, many interdependencies and large products, which take a significant time to complete at a substantial cost (Rolland et al. 2016). Dikert et al. (2016, p. 88) defined large-scale agile development as involving 'software development organisations with 50 or more people or at least six teams'. We use the term 'very large-scale agile development' to describe 'agile development efforts with ten or more teams' (Dingsøyr, Fægri, and Itkonen 2014). If each team has seven members, the project will involve 70 team members and will have the characteristics described above. In these projects, most of the challenges associated with scale become evident. We use the term 'programme' to refer to a collection of related projects.

There is a growing academic literature on large-scale agile development, after it appeared as a new topic in the discourse on agile development in the mid-2000s (Hoda et al. 2018). A literature review identified 191 studies, which were mostly experience reports (Edison et al. 2021). The review shed light on the underlying reasons for the interest in large-scale agile development and the need for alignment and cohesion across many teams, interdependencies between software development and other organizational functions, and the trend towards product delivery at scale.

In the special issue on large-scale agile development in *IEEE Software*, Dingsøyr et al. (2019b) described two waves of development methods.[1] We think that referring to these waves as the first and second generations of large-scale agile development methods is conceptually clearer because generations represent a more fundamental change that keeps living when the next generation arrives, while waves are short-lived.

Early advice on agile methods indicated that they are best suited for co-located teams developing software that was not safety critical (Williams and Cockburn 2003). For larger development efforts, Boehm and Turner (2003) recommended balancing traditional[2] and agile development methods.

2.1.1 First-Generation Methods

First-generation large-scale agile development methods combine agile methods at the team level with traditional project management frameworks, such as the Project Management Body of Knowledge (Duncan 2017) or Prince2 (Bentley 2010). Many refer to these methods as hybrid approaches (Bick et al. 2018). Project management frameworks enable a wrapping on the development process using traditional engineering approaches. This can serve as an interface to a more traditionally minded organization or customer. The frameworks are process centric, rely on formal communication and individual roles, divide work into phases like in the waterfall model and are oriented towards a bureaucratic organization (Nerur et al. 2005).

An example is the first published case study on large-scale agile development, which showed a combination of the Project Management Body of Knowledge with the agile method Scrum (Batra et al. 2010). This project for an American cruise company had a final cost of USD 15 million and involved 60% changes in requirements during execution, but it was still able to deliver in terms of time, cost and quality. The study pointed out the need for structure in the project management framework because the project was large, strategically important, time critical and distributed, while the combination with agile methods was necessary to handle unforeseen events and changes in requirements.

Another example showing how a model inspired by Prince2 was combined with Scrum is a Norwegian State Pension Fund programme with a total cost of around EUR 140 million. The programme was organized into four main projects: an architecture project, a business project, a development project and a test project. At most, 12 development teams worked in parallel, with the releases organized into the phases of needs analysis, solution description, construction and approval (Dingsøyr et al. 2018b). A team would often work on three releases in parallel, one under approval, another under construction and a third being planned. Scrum practices were followed at the team level, such as sprint planning, daily meetings, sprint backlog and team retrospectives. Demonstrations were held every three weeks in one meeting for all teams. The programme developed around 2500 user stories, organized into about 300 epics and with 12 releases.

[1] The term 'wave' is also used in white papers by practitioners like Charlie Rudd (accessed April 2022:https://www.solutionsiq.com/resource/white-paper/the-third-wave-of-agile-2/) and Steve Denning (Accessed April 2022: https://www.forbes.com/sites/stevedenning/2017/02/10/beyond-agile-operations-how-to-achieve-the-holy-grail-of-strategic-agility/?sh=5167e5982b6a) but then focusing more broadly on agility and not large-scale agile development methods.

[2] Some, such as the Agile Alliance and Project Management Institute's 'Agile Practice Guide' use the word 'predictive' rather than 'traditional'.

2.1.2 Second-Generation Methods

In recent years, we have seen what we call a second generation of large-scale agile development methods, in which much of the advice from project management frameworks is replaced by lessons learned from digital product development. These methods include SAFe, Scrum-at-scale, Disciplined Agile Delivery, LeSS, and the Spotify model (Dingsøyr et al. 2019a; Edison et al. 2021). In contrast to first-generation methods, these approaches embrace ideas from the agile community and bring in new insights from lean product development. They focus more on the product than the process, making greater use of informal communication, an evolutionary delivery model and an organic organization to encourage *cooperative social action* (Nerur et al. 2005). The management style is more oriented towards collaboration. The methods define principles built on ideas in the agile community (Baham and Hirschheim 2021) and prescribe the organization of large projects by relying mainly on teams; release planning and architecture through roadmaps and guidelines; collaboration with customers by involving them or end users at different levels; and typical practices for inter-team coordination, such as scrum of scrums meetings, and for knowledge sharing, such as communities of practice.

As an example, the multicase study of introduction of SAFe in the global telecommunications company Comptel (Paasivaara 2017), describes practices at team, program and portfolio level. The study describes organization of work as planning with epics on portfolio level, where tasks were given to programs, called "agile release trains". Development was done in "product increments". Each increment started in the cases with a two day planning session, which was followed by development for 10 weeks. There were new roles at this level, such as product manager, system architect and release train engineer. The release train engineer prepared and led product increment planning, Scrum of Scrum meetings and "took care of the improvement items and metrics" (Paasivaara 2017, p. 4). Teams adopt an agile method as Scrum or Kanban, and in the cases worked in two-week iterations. Of two cases studies, one had 14 teams and the other 12 teams. There were also two platform teams serving both cases. Teams were cross-functional with 5–10 members. There were regular community meetings between product owners.

2.2 Coordination

Why is there a need to coordinate? A widely used literature review on coordination studies describes coordination as the organizational arrangements that allow individuals to 'realise a collective performance' (Okhuysen and Bechky 2009). Collaboration and communication are considered indispensable in coordination but are separate concepts. We subscribe to this understanding of coordination in the following, but we will use Malone and Crawston's (1994, p. 90) definition of coordination as the 'management of dependencies'.

An analysis of dependencies in agile development teams resulted in a taxonomy with three main groups: knowledge, process and resource dependencies (Strode 2016).

- *Knowledge dependencies* are defined as the pieces of 'information required for a project to progress' and include knowledge about requirements, expertise (technical or task knowledge), historical knowledge about past decisions and knowledge about task allocation (who is doing what).

- *Process dependencies* are defined as 'task[s that] must be completed before another task can proceed', including activities and business processes.
- *Resource dependencies* occur when 'an object is required for a project to progress'. Examples are the availability of a resource (person, place or thing) and technical dependencies, such as interactions with another technical component in the software system.

Dependencies are managed through *coordination mechanisms*. Mintzberg (1989) identified direct supervision and standardization of work, outputs, skills and norms as central coordination mechanisms.

Coordination in organization research initially focused on static mechanisms in well-predictable environments. The dynamic aspects of coordination were described as mutual adjustment mechanisms—coordination based on feedback. Several scholars have criticized an overly static view of coordination and proposed a dynamic understanding of it (Okhuysen and Bechky 2009). Jarzabkowski et al. (2012) suggested a model in which the absence of coordination leads to the creation of new patterns of coordination which are stabilized. Given the focus on flexibility in work processes, changes in requirements and technology, software development is a field in which coordination is likely to be very dynamic (Dingsøyr et al. 2018c).

We elaborate on how we coordinate through a coordination model, describe traditional and agile approaches to coordination and then further describe agile approaches for small and large-scale projects. Next, we present findings to date on three modes of coordination in large-scale agile development.

2.2.1 How do we Coordinate?

Strode (2012) presented a coordination model in small-scale agile software development projects based on the previous work by Espinosa et al. (2007) (see Fig. 1). We adopt this model for large-scale coordination, with a focus on inter-team coordination instead of coordination within teams.

Coordination effectiveness is one of the many factors contributing to overall project success. Effectiveness is defined as the 'state of coordination achieved in a project given the execution of a particular coordination strategy' (Strode et al. 2012, p. 1233) and encompasses implicit and explicit components. The implicit component is based on the literature on

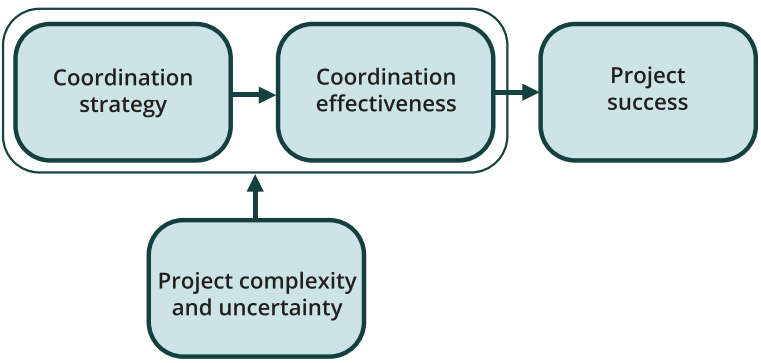


Fig. 1 Coordination strategy, coordination effectiveness and influence by project complexity and uncertainty (model from Strode (2012)

teamwork and coordination. It requires that project members understand the overall project goal and how tasks contribute to its realization, the overall idea about the project's status, the tasks to work on, the tasks that others are working on and where expertise is located in the project organization. The explicit component is that persons and artefacts are in the correct place at the correct time 'and in a state of readiness for use from the perspective of each individual involved in a project' (Strode et al. 2012, p. 1233).

How can we tell if coordination is not working? The late discovery of dependencies can lead to rework, for example, when integrating components from several teams and realizing that a new feature in one module is causing unexpected errors in another. Other problems could be due to several teams working simultaneously in the same part of the code base, which causes many merge conflicts in the code and could have been avoided if one team had delayed working in this part. There could be challenges with alignment, that some individuals or teams work on low priority tasks. If coordination is working well, it should be evident in constant progression on work tasks, unless there are other obstacles to progress. However, if a project invests too much in coordination, coordination mechanisms could be perceived as requiring too much time. If team members complain that specific meetings are not useful, this could signify a too heavy investment in coordination. Nevertheless, it could also be that meetings are not managed well and do not work effectively as coordination mechanisms.

The *coordination strategy* involves selecting a group of coordination mechanisms that manage dependencies in a situation (Strode et al. 2012). We use the term 'coordination strategy' more strictly than Berntzen et al. (2021) did, who described autonomous teams and technical architecture as strategies. We define coordination mechanisms in line with Van de Ven (1976), who identified three broad modes of coordination mechanisms:

- *Group mode* – mutual adjustment based on new information through feedback in meetings that can be either scheduled or unscheduled
- *Personal mode* – mutual adjustment through feedback but between two people at the same organizational level (personal, horizontal) or at different levels, such as a developer and a subproject manager (personal, vertical)
- *Impersonal mode* – use of 'codified blueprints of action', such as those in 'pre-established plans, schedules, forecasts, formalised rules, policies and procedures, and standardised information and communication systems' (Van de Ven et al. 1976, p. 323)

Choosing a coordination strategy involves finding a good set of coordination mechanisms that correspond to a project's *complexity* and *uncertainty* in a given situation. When describing a situation, we use the characteristics that determine coordination mechanisms, as Van de Ven et al. (1976) argued:

- *Task uncertainty* – This is the 'difficulty and variability of work undertaken by an organisational unit. Higher degrees of complexity, thinking time to solve problems, or time required before an outcome is known all indicate higher task uncertainty' (Dingsøyr et al. 2018c, p. 66).
- *Task interdependence* – This is defined as 'the extent to which people in an organisational unit depend on others to perform their work. A high degree of task-related collaboration means high interdependence' (Dingsøyr et al. 2018c, p. 66)

- *Size of the work unit* – This refers to 'the number of people in a work unit. Increases in participants in a project or program mean an increase in the size of the work unit' (Dingsøyr et al. 2018c, p. 67).

2.2.2 The Traditional and Agile Approaches to Coordination

Agile software development methods are designed to cope with change and uncertainty in small teams. They 'de-emphasise traditional coordination mechanisms such as forward planning, extensive documentation, specific coordination roles, contracts, and strict adherence to a pre-defined specified process' (Strode et al. 2012, p. 1222). Instead, they rely on synchronization through activities and artefacts, structure through proximity and substitutability of team members, and boundary spanning across teams (Strode et al. 2012). Table 1 shows the key differences between the traditional and agile approaches to coordination.

Pries-Heje and Pries-Heje (2011) attributed the success of the agile method Scrum to its flexible and efficient coordination structures. Agile methods also seek to move decision authority to the team level and rely on rough long-term plans and detailed short-term plans to increase adaptability to change (Xu 2009). This impacts who handles the components of the coordination strategy. In a position paper, Dingsøyr et al. (2018a, p. 82) stated that 'the complexity of large-scale agile development calls for rethinking coordination, emphasising characteristics such as oral communication, work in teams, a high level of interdependencies, uncertainty in tasks, many people involved, many relations between individuals and that coordination needs change over time'.

2.2.3 Coordination in Agile Development from the Small to Large Scales

An agile development team typically consists of five to nine members who work full time and are co-located. Boehm (2002) described the "home ground" for agile methods as smaller teams and products, seeking to provide rapid value in a context where refactoring is inexpensive and requirements may change rapidly. There are few communication channels in this context, and much of the management of dependencies can be done through feedback. Such feedback can go through the personal mode either directly between two team members (personal, horizontal) or in the whole team through scheduled meetings, as defined in Scrum (Schwaber and Beedle 2001). These meetings include daily iteration planning, iteration review and iteration retrospective meetings. Alternatively, the feedback can be given through unscheduled meetings, such as an extension of the daily meetings if these meetings identify an obstacle to project progress. Sharp and Robinson (2007, 2010) explained coordination in agile development as making collaboration easy because team members are very aware of others' work, the overall project progress, and the state of the code base. They identified two key artefacts for

Table 1 The traditional versus agile approaches to coordination (adapted from (Strode et al. 2012)

Traditional	Agile
Forward planning	Synchronization through activities and artefacts
Explicit documentation	Face-to-face communication (proximity)
Roles	Autonomous teams
Pre-defined processes	Boundary spanning across teams

coordination and collaboration: story cards with a description of work tasks (typically in the form of a user story) and a physical board that shows the work status in the current iteration. A study of artefacts used in the coordination of agile teams shows additional artefacts not described in agile development textbooks, such as a textual description of the business case, the contract and a wireframe mockup in the early development stages (Zaitsev et al. 2020) (see Strode (2012) for an in-depth discussion of coordination at the team level).

As projects grow in size, there will likely be more dependencies to manage. A study on teams' coordination needs in large-scale software development projects found that project-, team- and task-related characteristics impact teams' coordination needs. The satisfaction of these needs seems to influence teams' performance (Sablis et al. 2021). In another study, the coordination practices within and between Scrum teams were described as positively impacting delivery predictability in large projects (Vlietland et al. 2016). In globally distributed software development teams, Stray and Moe (2020) found significantly larger team sizes than those of co-located teams, and people working in distributed teams spent somewhat more time in meetings per day. A quantitative analysis of 71 SAFe projects from the company Rally found that dependencies were explicitly declared for about 10% of user stories (Biesialska et al. 2021). These dependencies were indicated in a lifecycle management tool by product owners, scrum masters or developers. The study emphasized that 'the volume of unidentified dependencies is not known' in the analysis (Biesialska et al. 2021, p. 27). Another study on several large-scale projects found that team members, on average, spent 1.1 hours a day in scheduled meetings and 1.6 hours per day on ad hoc communication and unscheduled meetings (Stray 2018). A finding from the exploratory study of the Perform programme with 12 development teams (Dingsøyr et al. 2018b) was that several unforeseen dependencies had to be managed, although the technical architecture and work organization were considered to minimize dependencies between teams.

Studies of multi-team systems in which many teams work together to solve larger tasks indicate that intra-team coordination (within teams) is vital for coordination between teams (inter-team coordination) (Firth et al. 2015). However, for teams' overall performance, inter-team coordination is most important (Marks et al. 2005).

2.3 Inter-Team Coordination

Inter-team coordination is a topic that has been given much attention in the literature on large-scale agile development (Bass 2019; Bjørnson et al. 2018; Dingsøyr and Moe 2013). A survey on coordination in large-scale software teams found that respondents hoped for more effective and efficient communication (Begel et al. 2009). The challenges identified across existing large-scale agile development methods are described in a systematic literature review on large-scale agile methods (Edison et al. 2021). These challenges include synchronizing across dynamic and fast-moving teams, addressing meeting overload (communication overload, external distractions), decreasing the many handovers between teams as a result of end-to-end development and maintaining transparency across a high number of teams.

Coordination challenges are shown in a case study of a large-scale hybrid development programme with 13 teams in a large enterprise software house (Bick et al. 2018). This programme had participants from three countries but did not find distance or sociocultural differences to cause challenges. An example of a challenge was that development teams' progress was blocked by unforeseen events, 'most frequently caused by an unidentified dependency with another team' (Bick et al. 2018, p. 939). Teams were often unaware of other

teams' activities, and team representatives were also not part of discussions on inter-team dependencies, as these happened in a central team that mainly consisted of people with business competencies. The study explained that the lack of dependency awareness between inter-team and team levels is rooted in misaligned planning activities during work specification and later prioritization, estimation and allocation of work to a team. Based on a rich data collection process, the study developed two propositions: i) dependency awareness is necessary but not sufficient for effective coordination, and ii) planning alignment of all phases is necessary but not efficient for dependency awareness. A recommendation for practice is to ensure regular inter-team meetings by using counterparts of standard team-level arenas for coordination in agile methods through joint planning, review and retrospective meetings.

A review by Edison et al. (2021) identified a set of practices across different large-scale agile development methods. Table 2 shows the practices relevant to inter-team coordination, which we grouped by coordination mode. In the following, we show knowledge to date on the group, personal and impersonal modes of coordination. Note that a recent case study on inter-team coordination mechanisms offers an alternative taxonomy, categorizing mechanisms according to the four characteristics of technical, organizational, physical, and social (Berntzen et al. 2022). We have chosen to use the modes proposed by Van de Ven (1976) to easier relate to previously published theory on interteam coordination.

2.3.1 Group Mode Coordination

The previous section's recommendation on regular inter-team meetings from the large-scale hybrid development programme builds on earlier studies on the scrum of scrums as an inter-team group mode coordination practice. A multicase study of large programmes with more than 20 development teams indicated that this area was not working very well. The topics discussed were not sufficiently relevant to the participants (Paasivaara et al. 2012). A recommendation was to downscale this forum to ensure the relevance of topics. This form of

Table 2 Inter-team coordination mechanisms from (Edison et al. 2021), organized by coordination mode (Van de Ven et al. 1976)

Mode	Mechanism
Group	Ad-hoc communication
	Cross-team demo (review)
	Mid-sprint review
	Physical proximity of teams
	Product owner coordination meetings, scaled product ownership
	Scrum of scrums meetings
	Synchronized sprint cycle
	Theme review meetings
	Virtual standup meetings
Personal	Ad-hoc communication
	Iterative proxy collaboration
	Team member rotation
Impersonal	Central team directives
	A collaborative platform
	Common goal for the sprints
	Regular full integration of software, hardware and mechanics
	A strategic roadmap
	Visualization (dependencies, deliveries and IT project portfolio)

scheduled meeting was also examined in the context of SAFe, with varied meeting outcomes. Two cases focused on status reporting and less on what is recommended in SAFe—to address risks. In one case, Gustavsson (2019, p. 9) reported that 'none regarded the meeting as a place to solve dependency issues', while in a third case, the meeting—based on the dependencies between teams—was used to help other teams. A misalignment between the corporate culture and coordination routines is suggested to explain the mismatch between intention in SAFe and practice.

Other studies have also shown that more meetings are used to coordinate large-scale projects. A survey and case study described large agile projects as having multiple 'committees of specialists, including the meeting of scrum masters in the scrum of scrums' (Hobbs and Petit 2017, p. 14). The study of the Perform programme found 13 coordination meetings, which were mainly scheduled meetings, including a joint demonstration and scrum of scrums meetings (Dingsøyr et al. 2018c). Retrospectives were, however, at the team level, but minutes from meetings were read and acted on by programme management.

A particularly interesting meeting in SAFe is the product increment planning meeting, described as a face-to-face event intended to create a shared mission and vision. Typically, the planning horizon is eight to twelve weeks, which is commonly divided into four iterations. Gustavsson (2019, p. 3) described this meeting as not only focusing on planning and highlighting dependencies but also 'inform[ing] and clarify[ing] the current context in terms of the business, product, and architecture'. The standard agenda in SAFe gives the most room for presentations, but a finding in three cases studied was that more and more time was used in team breakout sessions.

Another line of studies mainly involving scheduled meetings deals with aligning work by setting up groups for knowledge exchange across teams; these are called *communities of practice*. We find reports of how this practice is used in organizations, such as Ericsson (Paasivaara and Lassenius 2014) and Spotify (Smite et al. 2019), with insight into topics which are usually covered, such as agile methods, infrastructure and back-end and front-end development. Some communities focus primarily on learning or organizational development, while others have a more direct focus on coordination through standardization practices, for example, in defining coding standards or giving toolset recommendations (Smite et al. 2019). At Ericsson, these communities are described as having a critical role in the transition to agile development methods (Paasivaara and Lassenius 2014).

The group mode of coordination in large-scale agile development was further analysed in a study of two empirical cases, with a focus on changes in coordination modes over time (Moe et al. 2018). These changes included transitions from scheduled to unscheduled meetings and from unscheduled to scheduled meetings. The study concluded that programme management needs to be sensitive to changes in coordination needs over time. Edison et al. (2021) also identified unscheduled meetings as a practice, described as ad hoc *meetings* and *physical proximity of teams* in Table 2.

2.3.2 Personal Mode

The personal mode is used extensively in agile methods at the team level with pair programming practice. However, what do we know about using the individual personal mode of coordination in existing studies of large-scale agile development? Bick et al. (2018) described coordination at the inter-team level as mainly traditional, relying on roles and hierarchy. Although not reported, the personal mode was probably used for intra-team coordination

within teams through practices such as pair programming and possibly through vertical layers in the programme organization through direct communication between central team members and team roles, such as product owners. Issues were escalated from the team to the inter-team level, which could be an example of the vertical personal mode of coordination. In the Perform programme (Dingsøyr et al. 2018b; Dingsøyr et al. 2018c), horizontal coordination was facilitated by several factors, such as being located in the same physical open work area, which allowed for easy direct communication (ad hoc communication in Table 2), rotating team members between projects and forming new teams by splitting an existing team; several arenas for informal communication, such as lunches and coffee breaks, also existed. Pair programming was used extensively but mainly within development teams. Customer representatives were available in the open work area for consultation. The study reported that team members asked for advice across the teams and organizations which staffed subprojects, and many emphasized the important facilitating role of the open work area. Edison et al. (2021) also identified *proxy collaboration*, which we interpret as a role between teams that fits into the personal mode.

2.3.3 Impersonal Mode

Impersonal coordination in large-scale programmes was reported by Bick et al. (2018) as involving top-down planning, resulting in themes in a product backlog, epics in a release backlog and user stories and tasks in sprint backlogs. A similar masterplan was used in the Perform programme (Dingsøyr et al. 2018c) with deliverables that consisted of epics, which were again broken down into user stories and tasks at the iteration level. Table 2 lists the 'common goal for the sprints' and the 'strategic roadmap' (Edison et al. 2021).

We also find several descriptions of routines in the Perform programme, such as architectural guidelines, team routines and cross-team routines (e.g. scrum of scrums meetings). Furthermore, the planning is done more in writing than what is common in agile development, with a written description of the needs analysis and a solution description available on a programme wiki. This could be seen as central team directives (Table 2), but the guidelines are regularly updated based on feedback from retrospectives or work in architecture and business projects. A post-project review evaluated the use of guidelines and found that some were defined too late and some were not followed, as teams perceived that they resulted in less flexibility; obtaining an overview was also challenging because of the number of guidelines in the wiki. Furthermore, an instant messaging tool was used for asynchronous communication amongst all programme participants.

A particularly interesting finding from Perform is that the plan was both available in an issue tracker and as a physical board next to the team tables in the open work area. An informant stated, 'It takes two seconds to get an overview of the status [in a team], and from my location [in the open work area], I could see almost all the boards, and then I would know what had happened at the end of yesterday [in each team]' (Dingsøyr et al. 2018c). The study of inter-team coordination in SAFe cases showed variants of the board at the programme level. The programme board included information such as features, dependencies between features and relevant milestones for the next product increment (Gustavsson 2019). The study demonstrated the use of physical and electronic boards and the different frequencies of updates across the examined cases. Edison et al. (2021) listed several other studies that found visualization to be a common practice.

3 Method

To investigate our research question—*how is the inter-team coordination strategy impacted by a change from the first- to second-generation large-scale agile development methods*—we have designed a longitudinal embedded explanatory case study (Runeson and Höst 2009; Yin 2018). The systematic literature review of large-scale agile methods shows that 'purposefully designed longitudinal studies on the adoption and application of large-scale agile methods are rarely seen in the existing literature' (Edison et al. 2021). We draw on previously established theories on coordination, mainly from management science, and from prior studies of inter-team coordination in large-scale agile development. We position the study as a positivist case study seeking to explain the impacts of a change by drawing on prior theory to define a set of novel propositions. In the following, we describe the research design, the procedures for data collection and the data analysis. The main limitations are discussed in Section 5.5.

3.1 Case Study Design

The objective of the present study was to increase the understanding of coordination in large-scale agile development, particularly to empirically examine strategies for inter-team coordination. This means that we have not focused on coordination at the team level. Prior studies have identified changes in coordination mechanisms over time, but as Edison et al. (2021) found, few longitudinal studies have been conducted.

The case is a very large-scale agile development programme. A programme involves a temporal organization, which differs from a permanent software development organization in that many participants will work for a shorter period. The case was selected as one of several large-scale software development projects followed in a research project. The criterion for selecting the case was that it should be an extreme case for coordination in that it had a high number of development teams (what we describe as a very large-scale agile development programme) (Dingsøyr et al. 2014). The programme had 200 participants at the most, with about 130 working in 10 development teams and in programme organization. The programme was co-located, which meant that we did not need to focus on topics related to sociocultural distance (Ågerfalk and Fitzgerald 2006) or distributed agile development (Šmite et al. 2010). We describe the case as extreme for the following reasons. The first is its size. Second, the programme is also an extreme case of a large-scale agile development method in the initial choice of a first-generation large-scale agile development method which is more oriented towards plan-based development than, for example, the Perform programme (Dingsøyr et al. 2018b). Our case was more oriented towards plan-based development in that it had two projects, *business* and *development*, with formal handovers between them. When reorganizing, the programme chose to work with continuous deployment and autonomous teams, which we argue are more in line with agile principles (Baham and Hirschheim 2021) than some of the second-generation large-scale agile development methods that, for example, prescribe a number of roles.

The unit of analysis is inter-team coordination strategies between business and development projects in the programme. The original plan was to focus on how the programme adjusted its coordination strategies over time. The programme was planned with three releases, and the plan for data collection focused on documenting practices and perceptions of practices amongst different groups for each release. However, the programme did reorganize, which gave us a unique opportunity to study changes in coordination after reorganization. As a

consequence, we revised the data collection procedures, as described below. We focused on two phases of the programme in which 10 development teams worked in parallel: one phase using a first-generation large-scale agile development method and another phase using a second-generation large-scale agile development method.

We asked to follow the programme from early 2017 and were granted access to interview its participants, read relevant documents and observe meetings. We were also given a series of briefings about the organization and the progress of the programme.

The study was part of a more extensive work in which we already obtained approval from the Norwegian Centre for Research Data (reference 848,084). We secured informed consent from the interview participants and ensured that the data used in the reports are not traceable to individuals and that we regularly gave feedback about the findings to the case participants.

3.2 Data Collection

We had to carefully consider our strategy for data collection. The programme was located in Oslo, but most of our research team members were located 500 km away in Trondheim. We therefore chose to organize regular visits to the case, in which three to four researchers would participate in the data collection and subsequent discussion. A PhD candidate partly contributed to the data collection and gave us much insight into the context by studying changes in the central IT department of the case organization (Vestues 2021). The discussions after data collection were crucial in developing a collective understanding of the programme organization and coordination challenges amongst the research team.

Data collection was conducted through individual interviews, group interviews, observations and collection of documents. We also held meetings with programme management to obtain an understanding of the organization of the programme. Field notes were written after the meetings and observations.

We interviewed individuals in a variety of roles to understand coordination challenges and practices, as shown in Table 3. Our primary focus was on software development practices, and most of our informants had roles related to development; however, we also interviewed several individuals in other roles to understand programme organization. The interview guides were revised from a previous study (Dingsøyr et al. 2018b; Dingsøyr et al. 2018c) (see Appendix 1). These guides focused on coordination challenges and practices, as well as on contrasting between work on releases. The questions were mainly open and phrased in a language familiar to the respondents, such as 'What dependencies do you have on other teams? Examples?' and 'How do you manage dependencies'? We made minor changes in the last round of interviews

Table 3 Roles interviewed after the interview round and the phases in the programme

Phase	Roles interviewed
First phase – round 1	Application architects (2), construction responsible, developer (2), functional architects (2), functionality responsible, scrum master (2), senior solution architect, solution manager, tester
End of first / start of second – round 2	Customer manager, central IT, developer, functional architect, product owners (2), programme owner, scrum masters (3), testers (2)
Second phase – round 3	Application architect, central IT (2), developer (2), programme manager, project manager deployment, product owner (3), project manager development, solution architect, test automation, tester

to focus on the effects of work reorganization, which we call a transition from the first- to second-generation large-scale agile development methods.

We visited the case three times over two days. We were three to four researchers conducting semi-structured interviews in parallel, which were followed by a feedback session with our interpretation of what was said. During the visits, the first interviews were conducted by a pair of researchers to ensure consistency in the use of the interview guide. Later interviews were conducted by a single researcher. The interviews lasted from 24 to 120 minutes, typically around 30 minutes. These were recorded and transcribed for analysis. In total, we interviewed 39 informants—13 in December 2017, 12 in January 2019 and 13 in November 2019. We conducted another interview in January 2020 (see the participants' roles in Table 3). As described in the limitations section, we could not interview participants from all teams during all visits, but we always interviewed people involved in development or test, requirements engineering, architecture and project or programme management. In total, the interview material contained 456 pages of text.

We also invited key people from the programme to a workshop in October 2020, in which we established a timeline and brainstormed on what worked well and what could be improved. This workshop led to a separate article on key learning from the transformation process, written with practitioners from the case (Dingsøyr et al. 2022). We further conducted group interviews to discuss coordination and the requirements engineering process. The group interview on coordination included a project manager and a product owner from NAV and a project manager, an assisting project manager and the construction responsible for the development project from Sopra Steria. This two-hour interview was recorded and transcribed into a 42-page document.

When negotiating access to the case, we avoided data collection in periods close to a release. Consequently, the first round of interviews was conducted during a relatively calm period and could be characterized by a neutral mood amongst the subjects. The second round was done after the initial shock of the reorganization had settled, which was characterized by a mix of frustration and optimism. The third round was completed after the programme ended. One of the researchers wrote, '*I've never interviewed people who are uniformly so happy with their situation!*' (Field notes, interview round 3).

We observed arenas for inter-team coordination, such as daily meetings and planning meetings, when visiting the case. To obtain further insight, we also facilitated retrospectives on team coordination in November 2017 and one on the delivery model in January 2018. Apart from facilitating these two retrospectives, we did not intervene in how the programme organized inter-team coordination.

The documents included an initial overall plan (39 pages), the proposal to reorganize the programme (23 slides) and a document describing the new release pipeline (209 pages). We also obtained access to minutes from team retrospectives, which provided insight into what the teams perceived to work well and what was perceived as challenges.

3.3 Data Analysis

The data material was imported into a tool for qualitative analysis (Nvivo 12). All data material was anonymised, and files were given attributes that described the programme phase, role (where relevant) and which interview round the file belonged to (if relevant). The dominant data source used in the analysis was the qualitative interviews.

We used interview guides that gave us much context on the case. We first conducted descriptive and holistic coding on material related to coordination. Three researchers first coded the interviews independently and then compared the coding. This happened in a series of workshop meetings, and the goal was to align our understanding of the codes. The three researchers who participated in the coding all took part in the data collection and discussions of the case over time, and all had prior experience in coding similar material.

We further independently coded the material in more detail by using codes on coordination mechanisms, such as scrum of scrums meetings, issue trackers and artefacts, such as dependency maps. 22 codes were taken from previous studies on coordination in large-scale agile development (Dingsøyr et al. 2018b; Dingsøyr et al. 2018c). Coordination mechanisms were coded in broad groups using the coordination modes proposed by Van de Ven et al. (1976): the group, personal and impersonal modes. A sample text coded as 'scrum of scrums' and was related to the first phase of the programme was '… *we had scrum-of-scrums in which team leaders on each team met, and then we could raise issues with the other teams; we often identified if a team was waiting for another team, or if there were other causes for delay'*. We found 30 coordination mechanisms, as described in the results section.

We added coding about context, such as the descriptions of phases and product releases. The context information also included the codes used to describe 'programme complexity and uncertainty', 'perceived project success' and 'coordination effectiveness' (Fig. 1).

After coding, we engaged in several activities for within-case analysis (Eisenhardt 1989). We first generated reports for the coordination mechanisms for each phase, which were tabulated. Langley (1999) described this as a temporal bracketing strategy to theorize from process data in which we see fairly stable processes within each phase. We can then examine how the context affects each phase and determine the consequences of the processes in the form of coordination efficiency. We had several discussions within the research team regarding the findings, and we compared our initial results with those of another study (Carroll et al. 2020). Furthermore, the initial findings were presented, first, to the informants in the case and, second, in an online open meeting at the IT department. We also wrote a report in Norwegian, in which we presented the context and organization of the programme to obtain feedback on our understanding, and we developed a description for a narrative strategy (Langley 1999). Finally, in parallel with the analysis of the material for this article, the first author wrote a magazine article with the key participants from the case; the article summarized key learning from the transition (Dingsøyr et al. 2022). Overall, these activities helped us increase our understanding of the organization and the challenges in the case.

Through this iterative process (Eisenhardt 1989), we built an explanation of coordination in the case. Following the steps described by Sjøberg et al. (2008), first, we drew on existing constructs from coordination theory from Van de ven et al. (1976) and Strode et al. (2012), together with constructs from software engineering and agile software development. We also used our novel definitions of first- and second-generation large-scale agile methods. Second, by contrasting the two phases in the case study, we developed five novel propositions on coordination in large-scale agile development, which we suggest describe the impact of coordination in the transition from the first- to second-generation agile development methods. Third, the discussion shows our logical justification for the proposition, building on both our interpretation of the case study and our synthesis of related work presented in the background section. Fourth, we discuss the scope of the suggested propositions in Section 5.4. Finally, we discuss how the propositions might be tested in Section 5.5.

Table 4 Roles on programme level and roles in development teams in the first phase

Roles on programme-level	Roles in development teams
Construction manager (1), controller (1), customer manager (1), environmental manager (1), functional architects (7), functionality responsible (1), performance test manager (1), project manager (1), project support (1), PMO (1), quality assurance (1), solution manager (1), senior solution architect (1), test automation (1), test data responsible (1), test manager (1).	Scrum master (1), application architect (1–2), developers (5–6), testers (1–2)

4 Results

We first describe the parental benefit programme with its background and main objectives. The presentation of the programme is built on analysed documents, external media coverage and descriptions from informants. Section 4.2 describes the first phase with the organization of the programme into projects (Fig. 4) and roles (Table 4); it presents findings on the effectiveness of coordination in this phase, followed by findings on coordination mechanisms. Similarly, Section 4.3 describes the second phase with programme organization with autonomous teams (Fig. 7) and competence needs (Table 7), followed by findings on coordination effectiveness and coordination mechanisms.

4.1 The Parental Benefit Programme

As part of the welfare system in Norway, parents with newborn babies can apply for benefits as compensation for lost salaries during their parental leave. Every year, NAV processes about 100,000 applications for parental benefits or changes to these and distribute EUR 2 billion to parents.

Prior to the parental benefit programme, parents filed applications for parental benefits on a modern web interface. Then, NAV manually entered information from applications on paper into another interface to process the applications. These were then handled using IT solutions running on mainframe computers from the 1970s. NAV received 282,000 telephone inquiries from users on these benefits per year. The system was described in national media as 'complicated', 'time-consuming' and 'incomprehensible'.[3]

Overall, NAV runs more than 300 IT systems and operated with a model in which large programmes to modernize IT solutions were given to subcontractors. In 2012, they initiated a modernisation programme with a total budget of EUR 330 million to replace systems from the 1970s with a new platform with new services. Shortly after its initiation, 17 development teams recruited from five subcontractors worked in parallel. After nine months, the modernisation programme was stopped because of a lack of progress; the cost was about EUR 70 million. This led to a parliament hearing and the resignation of the IT director and the director of NAV. '*The trust from the ministry was totally broken*', one of our informants in the programme management stated (round 3).

The further modernisation of IT infrastructure was then replanned by smaller programmes seeking to reduce risk, building on known technology and development processes. The parental benefit programme was the second of three programmes, and the aim was to digitize

[3] https://www.vg.no/nyheter/innenriks/i/82rdG/nav-det-er-for-vanskelig-aa-soeke-om-foreldrepenger

the application process for new parents' parental benefits. Because of a new law, the old system was to be replaced by 1 January 2019.

The new solution aimed to reduce the number of inquiries by 25%, achieving a self-service rate of 80% and decreasing incorrect payments by 10%. NAV described the goal to be achieved as follows: *'(1) automatic application processing, (2) users can manage their application through the self-service solution and (3) electronic collection of information from caseworkers will provide better quality and more efficiency in application processing'*. (document describing the programme).

The parental benefit programme lasted from October 2016 to June 2019. We studied the main part of the programme, which, at its peak, employed 130 people[4] in 10 teams, of which 100 were external consultants from "Alpha" and Sopra Steria. The programme manager was employed by NAV. The programme depended on functionality in about 20 other systems at NAV.

The programme started by using an internally tailored first-generation large-scale agile development method similar to that used in the Perform programme at the State Pension Fund (Dingsøyr et al. 2018b), with certain changes. There were three planned releases—the baseline, the settler and the digital—all including 50,000 to 75,000 hours of estimated work. Nevertheless, for reasons that will be described in the following, the development model was changed to a second-generation method in October 2018. As shown in the timeline in Fig. 2, the programme started with one development team and gradually increased the number of participants to 10 teams, which we describe as a very large programme. We reported the lessons learned from the transformation process in a separate article (Dingsøyr et al. 2022). The whole programme was physically collocated in the same work area, as shown at one time in Fig. 3, on two floors. Some participants in the programme had also worked in the Perform programme and had a background in this development method. The programme used a target price contract model (PS2000 SOL) for the first two releases, but this was changed to a time and material model in the second phase.

4.2 First Phase

The first phase included two releases. The baseline release was a digital application processing system that automatically processed applications for one-time benefits. The settler release expanded the application processing system to include all types of parental benefits and integration with employers' pay systems. This phase aimed to develop a complete decision-making system adapted to the requirements of calculation in the law.

In this phase, the work was organized into four projects: business, development, test and change management (Fig. 4). The business project was responsible for the phase of *analysis of needs*, which was conducted in collaboration with the development project given a *solution description*, before being assigned to a development team in the *construction* phase; after development, the *approval phase* organized by the test project followed. This model was similar to what was used in the Perform programme (Dingsøyr et al. 2018b). The programme could then, at a particular time, be in the production phase of one release while being in the construction phase of a second release and conducting the needs analysis for a third (Fig. 5).

[4] The whole program had about 200 people at peak.

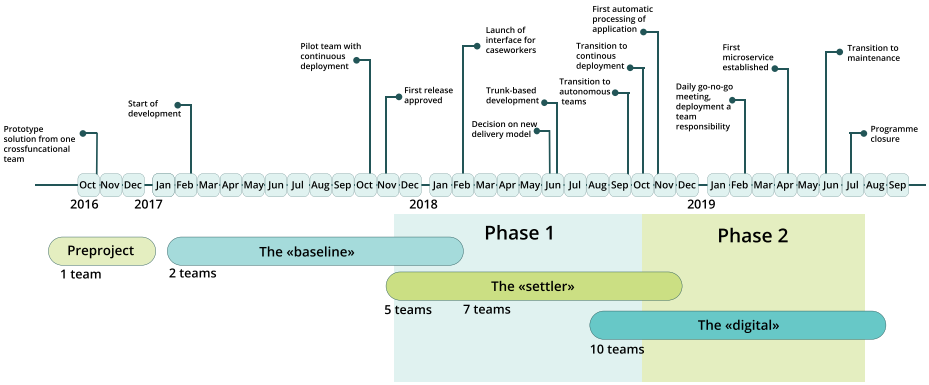


Fig. 2 Programme timeline

The change management project introduced new solutions to the main user groups, end users seeking parental benefits and caseworkers at NAV.

The development teams worked in three-week iterations with the four roles described in Table 4. The business project and the development teams were located in different parts of the work area, and the functional architects were located with the business project, but they prepared solution descriptions of user stories for the development teams. These were made in the programme wiki. There were 16 roles at the programme level, which are described in Table 4.

When starting on the second delivery (settler, Fig. 2), the programme created a pilot test to examine second-generation large-scale agile development methods in a cross-functional

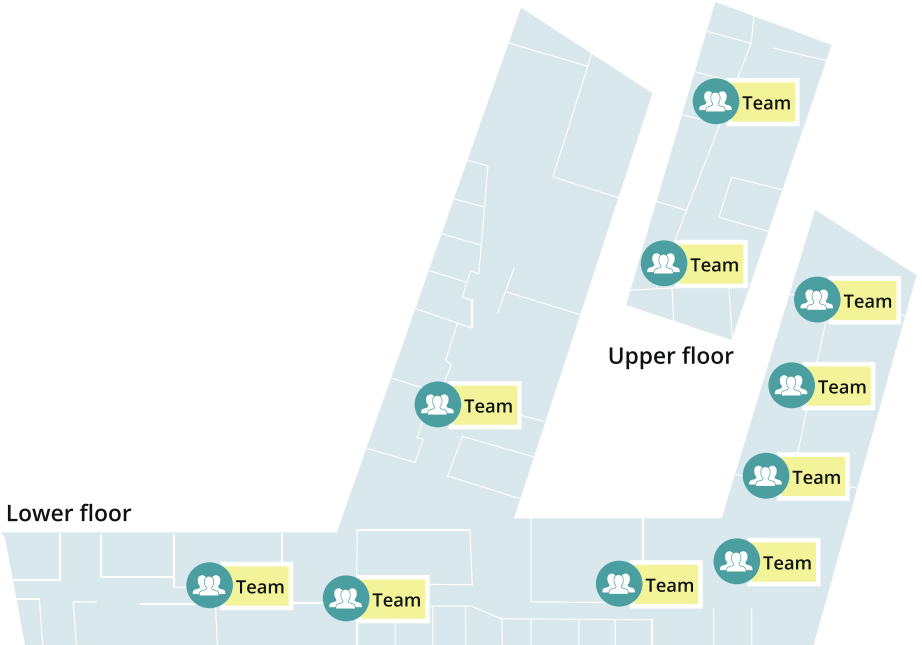


Fig. 3 Physical work area where the programme was located in both phases

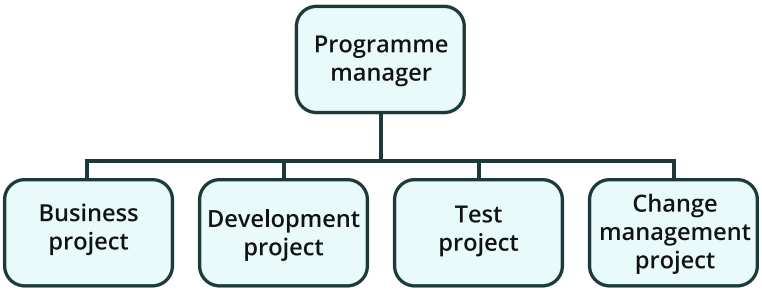


Fig. 4 Organization of the programme with four main projects

autonomous team. A committee was formed to assess whether the entire programme should change the delivery model.

In the focus group interviews, the informants described this phase as being characterized by not only time pressure but also a meeting culture in the programme. This made decision making time consuming:

> '*It was a constant pressure to deliver. We had six to seven development teams that should continuously be fed tasks for their sprints. And that is quite a number of people and quite a lot of power in consuming user stories*' (manager, development project, group interview).

> '*... people were in meetings the whole time, and you'd never find anyone by their desk; because you didnt' find a person there, you had to invite them to a meeting ... And when first inviting, you'd also invite more people to make sure*' (business analyst, business project, group interview).

An informant stated that, as people tended to have full schedules, calling for a meeting often would delay decision making by more than a week.

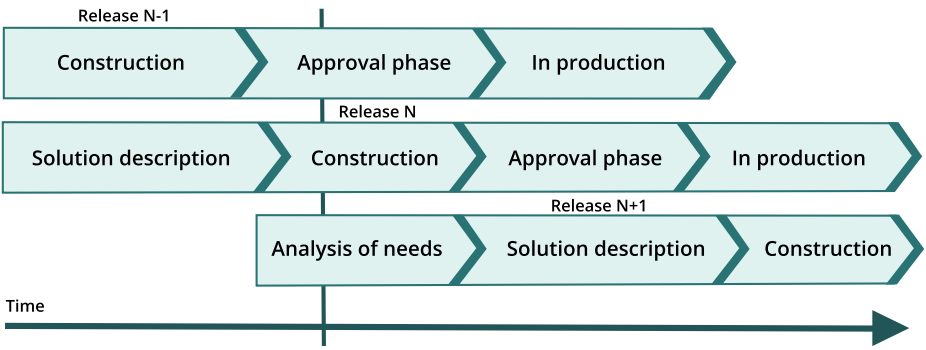


Fig. 5 Development phases

4.2.1 Coordination

The coordination in the first phase of the programme was characterized by the value chain, with formal handovers between the phases (Fig. 5).

NAV used the consultancy company "Alpha" to assist in creating solution descriptions. NAV and consultants from "Alpha" coordinated internally to prioritize and harmonize the requirements across the value chain (CI1 in Fig. 6). The solution descriptions were then handed to a group of consultants from the development project, who processed these into user stories; these had to be approved by NAV before they could be handed to the development teams (CE2). The development teams had to coordinate internally (CI2) in order to develop the necessary code in the construction phase before handing the results back to NAV for testing and approval. If the solution descriptions involved external systems, NAV or consultants from "Alpha" would initiate contact with external partners to clarify how the process could be done (CE1).

When the user stories are passed to the team level, the team would have to initiate new contact with external partners in order to coordinate and book the necessary resources for developing the external system (CE3).

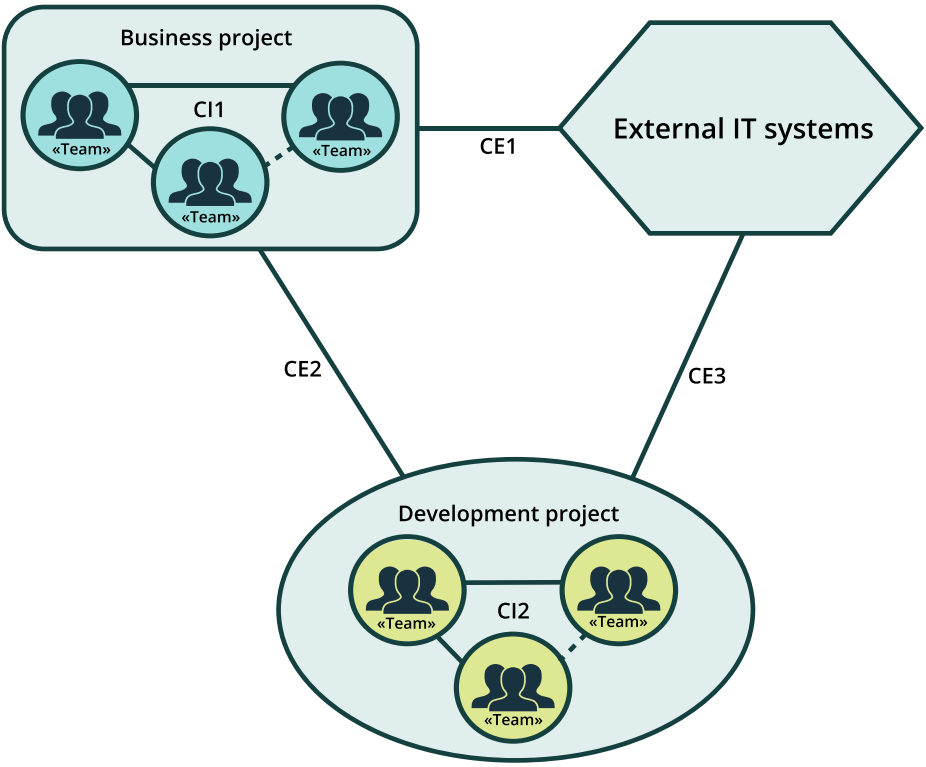


Fig. 6 Overview of coordination when using first-generation large-scale agile development methods. CI is the internal coordination in the programme, whereas CE refers to the various types of external coordination. Adapted from the whiteboard during group interview on coordination. The dashed line indicates that there are more than three teams

Interviews with key persons in the programme indicated that internal coordination was perceived to be working well:

> '*Coordination internally in the business project and internally in the development project worked well'* (manager, development project, group interview).

However, all parties expressed frustration with the coordination between the business project and the development project in the first phase (CE2):

> *'The coordination between projects was more demanding*' (manager, development project, group interview).

> '*In the business project, it was impossible to get insight into and obtain an understanding of what was happening and how they were working in development. You described needs, and it was like delivering to a black box*' (business analyst, business project, group interview).

A retrospective in January 2018 focusing on the delivery model identified the 'transitions between [the] phases [of] *analysis of needs, solution description* and *construction*' (Fig. 5) as a main challenge. In the following, we will more closely examine internal coordination in the development project, as well as the coordination between the business project and the development project. In total, we identified 27 coordination mechanisms for CI2 and CE2:

4.2.2 Inter-Team Coordination in the Development Project

Internal coordination between the development teams in the development project was highly structured. We identified 18 coordination mechanisms, as shown in Table 5, in which nine are group mode mechanisms, five are personal and four are impersonal. An iteration would start with a planning meeting in which the programme gathered all teams and presented tasks and dependencies for the upcoming iteration. The teams would then break out for individual team planning. Dependencies with other teams were mostly handled through the scrum master, who would contact the scrum master of the team which had the dependency. After contact was initiated, the developers involved would talk directly, use instant messaging or mail, or hold ad hoc meetings to resolve dependencies. Teams working closely in an iteration could also be moved physically next to one another to ease informal coordination.

> *'We did it periodically—moved people around. Teams 2 and 4, for example, often worked closely together, at least we used to in the last iteration, so then we moved together for a time'* (application architect, development project, round 1).

The scrum masters conducted a daily standup for their team. The standups were staggered, so it was possible to attend another team's standup if a team had dependencies that needed to be discussed. The scrum masters would also meet twice or thrice a week for a scrum of scrums meeting.

Each team had a technical architect who attended a technical architecture forum. The development project held what they called a technical review to transfer knowledge about

Table 5 Coordination mechanisms, classifications, descriptions and coordination modes for internal intra-team coordination in the development project (CI2)

Coordination Mechanism	Classification	Description	Mode
Demo meeting	Meeting	Attended by all teams	Group, scheduled
Functional architecture forum	Meeting	A forum to discuss dependencies	Group, scheduled
Planning meetings	Meeting	A formal meeting to initiate an iteration. All teams attended. Presented tasks and dependencies for the upcoming iteration.	Group, scheduled
Retrospectives	Meeting	Mandatory at the end of each iteration. All retrospectives collected in a common wiki.	Group, scheduled
Scrum of scrums	Meeting	Information flow between the teams. The project manager was the key stakeholder. Some discussion on dependencies.	Group, scheduled
Staggered standup meeting	Meeting	Mandatory and staggered so that other teams could attend one another's standup	Group, scheduled
Technical architecture forum	Meeting	A forum to discuss architectural dependencies	Group, scheduled
Technical review	Meeting	An internal forum for the consultancy company for knowledge transfer	Group, scheduled
Ad hoc meetings	Meeting	Used to clarify issues	Group, unscheduled
Team member rotation	Environment	Used often when the number of teams was growing to spread knowledge	Personal, horizontal
Pair programming	Meeting	Used to get graduates up to speed in the project	Personal, horizontal
Instant messaging	Tool	Mainly Skype and Hipchat	Personal, horizontal
One-on-one conversation	Meeting	Contact with other teams usually went through the scrum master.	Personal, vertical
Key roles	Role	Certain people were critical for keeping an overview and coordinating. Main architect, construction responsible.	Personal, vertical
Architectural guidelines	Artefact	Some overall figures available on Confluence	Impersonal
Dependency map	Artefact	Introduced after a while to give developers an overview of who they would interact with during an iteration	Impersonal
Physical layout	Environment	Teams were moved around when working on interconnected tasks	Impersonal
Whiteboard	Environment	All teams had their own.	Impersonal

new technology, and all developers could attend. This meeting was described as one of the most important ones for inter-team coordination. One participant stated, '*The technical review is very good for aligning technical development across the teams*' (minutes from the retrospective focusing on inter-team coordination in November 2017).

During the first phase, the development project scaled up by adding more people; once the teams grew too large, they were split, and new people were added. This led to what they called 'stirring the pot', and most developers were rotated between several teams, thus bringing domain knowledge with them. The development project also had some roles on top of its team structure; these were considered important coordinating roles. The construction responsible

was often mentioned as a role that was engaged in frequent discussions with the teams to ensure that the right people were coordinating across the teams:

> *'The construction responsible worked almost full time with tasks which were in between teams'* (manager, development project, the group interview).

At the end of the iteration, each team conducted a retrospective and documented the results in a wiki.

They also arranged a common demonstration in which each team showed internal and external stakeholders what it had produced in the iteration and sought to align demonstrations from the teams:

> *'We tried to achieve a flow there … we tried to talk about where we worked on in the solution and achieve a natural flow, and then we got a smooth transition to the next team'* (scrum master, development project, round 1).

Table 5 shows all the coordination mechanisms identified.

4.2.3 Inter-Team Coordination Between Business and Development Projects

Table 6 provides an overview of the coordination mechanisms between the business and development projects. In our material, we identified a total of nine mechanisms, four group modes, one personal mode, and four impersonal mode. For coordination between the two projects, the development project had a dedicated team of what they called functional architects, who would handle contact with the business project. The idea was that these team members would divide their time equally between writing user stories and being available for the development teams that would implement the user stories to clarify issues. In practice, they spent most of their time in meetings with NAV. User stories were specified in formal and informal meetings. There were formal working meetings to initiate work on a user story, and there could be several user story meetings between the functional architects and the business project to clarify issues. Finally, there was a formal approval meeting with NAV before the user story was transferred to the business project's issue tracker and scheduled for a future iteration.

> *'Regarding the solution descriptions, there were several meetings … both internal to us and with the customer to work on those'* (project manager, development project, group interview).

Many informants stated that a major challenge with coordination in this phase was that the teams working on solution descriptions and user stories and the teams developing the solution were not working on the same user stories simultaneously.

There was a perception of time pressure in the programme. A functional architect (development project, round 1) stated that '*The deadlines are short … we need to deliver to the approval meeting on Thursday afternoon, have the approval meeting on Friday afternoon … That is not how I'd like to do it'*. Construction would then start the next week.

It could take months from the approval of a user story until a team began implementation, and if there were issues that needed clarification, the people who wrote the description worked

Table 6 Coordination mechanisms, classifications, descriptions and coordination modes for the coordination between the business and development projects

Mechanism	Classification	Description	Mode
Approval meeting	Meeting	Formal meeting in which NAV approved the user story	Group, scheduled
Demo meeting	Meeting	Attended by all teams	Group, scheduled
Working meeting	Meeting	Formal meeting between the functional architect and the business project to start writing a user story solution description	Group, scheduled
User story meeting	Meeting	Informal meetings to clarify issues in user stories between the functional architect and the business project	Group, unscheduled
Functional architects	Role	Dedicated team of people from the development project responsible for detailing user stories together with NAV and clarifying issues from developers	Personal, horizontal
Dependency map	Artefact	The functional architects made a map of dependencies between user stories and presented it at the beginning of every iteration.	Impersonal
Issue tracker	Artefact	Jira was used to transfer user stories to developers.	Impersonal
User story	Artefact	The approved solution descriptions for user stories were transferred to the development project's issue tracker and then scheduled for a future iteration.	Impersonal
Physical layout	Environment	Developers separated from solution description; functional architects close to developers	Impersonal

on new tasks and had to try to recall what they had meant. This also led to a long feedback loop and limited learning across organizational lines. The functional architects had their own forums in which they discussed dependencies and tried to identify as many as possible before development began. After a while, they introduced a dependency map presented to the developers at the beginning of every iteration to increase awareness. Initially, the functional architects were placed together with the business project, but they were eventually moved into the development project with the teams they supported.

As stated, the retrospective in January 2018 focusing on the delivery model identified the 'transitions between the phases of analysis of needs, solution description and construction' as a main challenge, which included a 'too high focus on details early' and 'too late prioritisation of requirements'. An informant described that the '*documentation of needs and solution descriptions was very extensive*' and that '*requirements were very detailed*' (business analyst, business project). At this stage, other challenges identified in the retrospective were 'information flow across the programme' and 'too many and too long meetings'.

4.3 Second Phase

The aim of the last release, the digital, was to create a self-service function integrated with an extended application processing system and to support integration with health actors. The goals for the release included creating a complete integration between a planning calendar and a dialogue about benefit applications with users and conducting a digital dialogue between the user and the application caseworker. The previous phase created a minimum viable product of core functionality that was to be further developed. A main difference of this release was that

the programme was now to develop a solution that was in use and add new functionality in a domain that was less well explored.

The programme manager set up an internal committee to suggest an organization and delivery model for the last phase. They were mandated to propose changes in working methods that could enable the programme to work better but which would not increase the risk for the previous phase or for the time when a new solution was to be released (document, proposal to reoganise the programme). Both NAV and suppliers were represented in the working group. At the end of the first phase, the team was allowed to work independently as an autonomous team that could continuously deploy new functionality. This team had good experiences.

Furthermore, the central IT function in NAV defined a new way of working that was different from the first-generation large-scale method of the first phase. A new IT director had a vision that all IT developments should be done with agile methods (see Mohagheghi and Lassenius (2021), (Bernhardt 2022) and (Vestues and Rolland 2021) for a description of changes in the IT department). A new technical platform was introduced in other parts of NAV, in which many non-functional requirements were handled in the platform; this made development teams focus better on functionality towards users. This platform used container technology and microservices and enabled an event-driven architecture.

Programme management did not think they *had* to change the delivery model: *'Given the size and complexity of the programme, it was well run—we delivered on time and we delivered on budget*'. However, they found that '*It was very calculated; yes, we had sufficient control so that we can work smarter. It was not like if we don't change now, we'll not deliver'* (manager, NAV, round 3). However, other informants expressed that delivering a more complex solution on a running system would have been challenging if the model had not been changed. A software architect (round 3) stated, '*We would not have had a chance*' to deliver a consistent solution without changing the model.

The internal committee proposed reorganizing to cross-funcational autonomous teams, with a gradual transition to continuous deployment (Fig. 7). The programme manager accepted the proposal, which led to significant changes in the last phase.

Some were worried about the transition to autonomous teams: '*I remember that at the beginning, people were worried about how we can keep oversight, how we should coordinate this and ensure that parts were coherent and that the teams align'* (business analyst, group interview).

The change was perceived as a fundamental transition:

> *'We were willing to adjust how we defined needs and solution descriptions. We've transitioned from one extreme point to the other, from massive models with areas, epics and user stories where everything is connected to the situation today, where things—in the best case—are documented in a Slack thread'* (team manager, group interview).

Informants stated that there *'was a lot less documentation … which I think everyone appreciated*' (business analyst, group interview), and *'a lot of roles disappeared*' (architect, group interview). The work tasks were more focused. One informant stated the following:

> *'The number of tasks you worked on simultaneously was reduced. But the quality of what was done was greater. Tasks used to take a long time previously, which led you to*

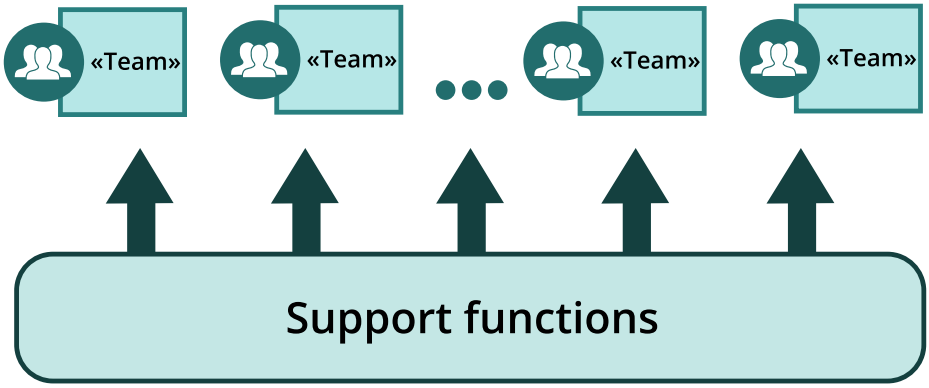


Fig. 7 The new organization with teams and supporting functions

have many tasks in process at all times. Now, the feeling was "this need will be delivered by the end of the week"' (business analyst, group interview).

There was an initial period characterized by a lack of coordination between teams:

'I don't know much about what the other teams are doing now' (test responsible, round 2).

However, the general perception was that it took time to adjust to the new delivery model, but eventually, *'we had a more streamlined use of tools, collaboration and coordination'* (manager, group interview). An informant stated, *'I found that we were providing a lot more value in production the last half year of the programme'* (business analyst, group interview). As we will describe, many of the old coordination mechanisms were re-introduced.

New regulations regarding the product were implemented in the winter of 2019. The product went into maintenance and further development in June 2019. The programme won the Norwegian prize for digitalisation the same year. Key objectives were met, such as the degree of self-service on applications which was higher than the target of 80% (99.8% in the spring of 2019). The time used to process applications was reduced from weeks or months to a matter of seconds.[5]

4.3.1 Programme Organization

The programme was now organized with 10 cross-functional autonomous teams for all product areas, as shown in Fig. 7. These teams were co-located and responsible for the product as a whole, including quality. The degree of autonomy was adapted to the degree of coupling between teams and dependencies. Still, most teams were eventually allowed to continuously put deliveries into production.

Development was now organized according to a flow-based model (Fitzgerald and Stol 2017), which resulted in the disappearance of roles in the programme, and new cross-functional autonomous teams were established with people from NAV and the two suppliers. Much thought was given to organizing teams according to the product domain in a way that would minimize the need for coordination.

[5] https://www.nrk.no/norge/na-kan-du-fa-svar-pa-fodselspengesoknad-pa-ett-minutt-1.13915937

Continuous deployment started in early 2019. Many meeting arenas disappeared. New support functions were established, as described in Table 7, and the teams received support from two agile coaches to further develop their work processes. They also received initial support from solution architects to ensure holistic architecture. The contract model was changed so that the suppliers delivered resources to NAV.

The autonomous teams were described as cross-funcational autonomous product teams and had approximately 12 members without formal management. Each team had a product owner. The teams were sometimes moved in the office landscape to sit close to the teams with which they collaborated.

Each team had a product owner from NAV; the consultants from "Alpha" and the functional architects from Sopra Steria were designated functionals and tasked with helping the product owner, as shown in Fig. 8. Otherwise, the teams mainly consisted of development teams from the first phase. Some developers from NAV were also integrated into the teams. These developers met across the teams and would eventually become the team that would take over the solution once the development programme had ended.

4.3.2 Inter-Team Coordination Between Autonomous Teams

According to our informants from both consulting companies and NAV, the coordination between NAV and developers improved with the new structure. *'It strengthens the developers' understanding of the domain and the product owner's understanding of the technology. You save a lot of time and get more work completed'*. (functional architect, group interview).

At the same time, most of the arenas across the teams were removed in the reorganization to allow the teams to be autonomous and freely decide on their involvement in meetings. Many teams saw the arenas as timeconsuming and not crucial when operating as an autonomous team. The first two to three months after the reorganization were challenging, mostly because the developers from Sopra Steria were still under the old contract to deliver the last big delivery. Once that had been delivered and the teams moved to daily deployment, the team members from NAV, "Alpha" and Sopra Steria got a more similar focus, developed an identity as a team and aligned their working processes.

All tools and processes were dropped in the reorganization, and teams adopted different approaches to how they would like to work. Some lifted the old process into the new team structure; others swore never to work with the wiki tool again. Eventually, some standardization and new meeting arenas emerged in the new team structure. However, teams started to take responsibility, and the need for competence at the programme level was quickly reduced (Table 7).

Table 7 Competence needs at programme and team levels

Programme level	Development teams
Benefit management, business management, coordination with "external" teams, environmental coordination, holistic architecture, project management, PMO function (adapted to new model), restructuring and communication, value chain testing across teams, UX - holistic design.	Development competence, domain and business competence, functional competence, team-lead competence, technical architecture, test competence, competence, UX competence.

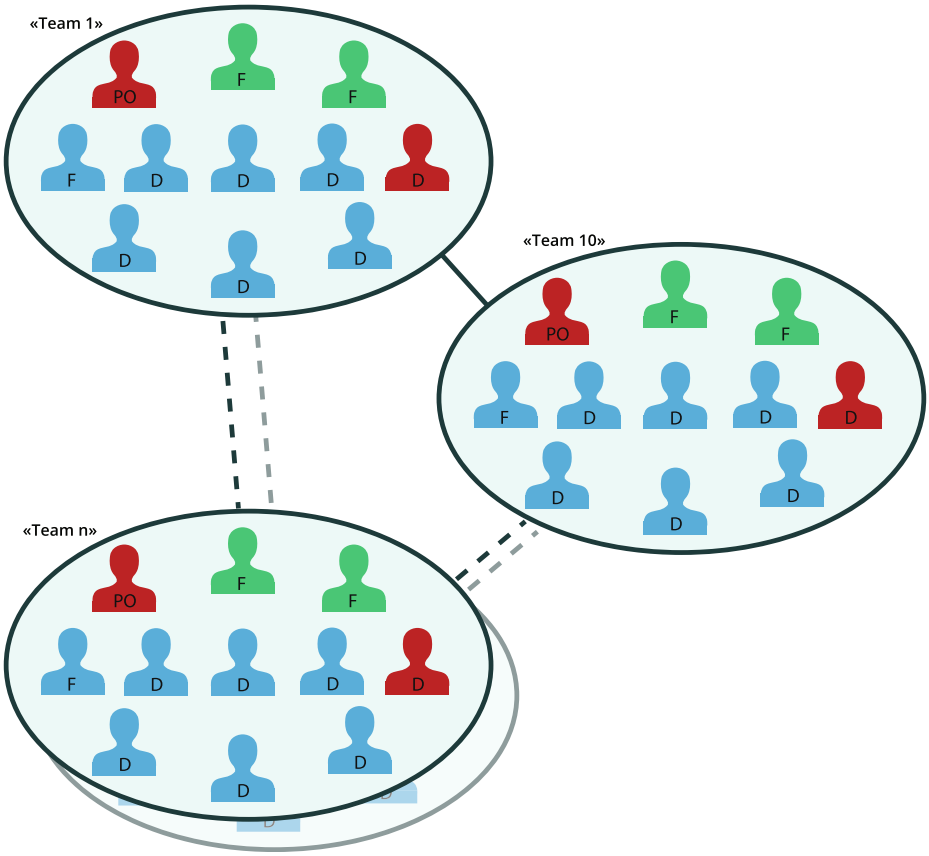


Fig. 8 The reorganization into autonomous teams led to more intra-team coordination and less inter-team coordination. Teams were cross-functional with product owners (POs) on each team, further team members who had formely had roles as developers (D) and functional architects (F). The participants from two consulting companies are shown in blue and green, participants from NAV in red. Teams would typically have 12 members

As shown in Table 8, we found 14 coordination mechanisms in this phase—seven in the group mode, three in the personal mode and four in the impersonal mode. Some mechanisms that reappeared did so in a different form, such as the demonstrations, which used to be scheduled meetings but were now unscheduled. One informant missed the scheduled demonstrations, which gave insight into what other teams were doing:

> *'… I miss that, but I see that it could be difficult with the teams being autonomous and they deciding what to show. So now, we have internal demos in our team; we try to have them weekly'* (test responsible, round 2).

A common repository was used to host all code, and the issue tracker was reintroduced as the standard way of documenting user stories, now including possible dependencies on other teams. The scrum of scrums meeting was reintroduced to handle dependencies between teams. The product owners reintroduced a product owner meeting to obtain a better overview of the total solution. Furthermore, the functionals reintroduced a forum across the teams to discuss dependencies between user stories, and the tech leads of the teams started meeting weekly to

Table 8 Coordination mechanisms, classifications, descriptions and coordination modes for inter-team coordination when using second-generation large-scale agile development

Coordination mechanism	Classification	Description	Mode
Functional architecture forum / status meeting	Meeting	A forum to discuss dependencies. Removed initially but reintroduced as a status meeting when they saw the need for it.	Group, scheduled
Scrum of scrums	Meeting	Information flow between the teams. Removed initially but reintroduced.	Group, scheduled
Go-no-go meeting	Meeting	Daily meeting in which a decision was taken on what to deploy	Group, scheduled
Techlead forum	Meeting	A weekly meeting with techleads from all teams	Group, scheduled
Technical review	Meeting	An internal forum for the consultancy company for knowledge transfer	Group, scheduled
Ad hoc meetings	Meeting	Increased use in this phase. Mainly used between similar roles across teams. Functionals experienced a large increase in these since the formal meetings were removed.	Group, unscheduled
Demo meeting	Meeting	Optional if a team calls a demo and invites participants from other teams	Group, unscheduled
One-on-one conversation	Meeting	No specific contact person on a team, so they had to talk to other people directly.	Personal, horizontal
Instant messaging	Tool	Mainly Slack; its use was increased significantly.	Personal, horizontal
Key roles	Role	Most of these roles were removed; construction responsible was still a key person.	Personal, vertical
Physical layout	Environment	Integrated and collocated teams	Impersonal
Whiteboard	Environment	All teams had their own.	Impersonal
GitHub	Tool	A development platform for hosting and reviewing code. Introduced when they began with daily deployment.	Impersonal
Issue tracker	Tool	Jira still in use, less details on requirements, dependencies were added.	Impersonal

discuss technical dependencies. A go-no-go meeting was introduced daily to discuss whether to push the code from the previous day to production, which was described as an important meeting that provided the participants with an overview of the programme's status. At the same time, the use of informal person-to-person or ad hoc meetings increased along with instant messaging.

The work tasks for developers were less specified, which meant that they had to discuss more with businesspeople. Some teams introduced a start-up conversation when initiating work on a task, which was guided by the following questions: What is this task? Why should it be this way? What are we looking for? The task description might be just a sentence or two.

To coordinate, the teams used task boards and an issue tracker with product queues for backlog refinement and a common roadmap with an outline for the next four months.

5 Discussion

How is the inter-team coordination strategy impacted by a change from the first- to second-generation large-scale agile development methods? The coordination strategy involves a choice of coordination mechanisms to achieve coordination effectiveness in a certain situation. Coordination effectiveness is an essential contributor to overall programme success. We start by discussing the differences in the programme's situation in the first and second phases. This is followed by the perceived coordination effectiveness in the first and second phases. Finally, we discuss the differences in the corresponding coordination strategies and suggest five propositions related to our research question before discussing the main limitations.

5.1 Changes in the Programme Situation

To describe the situation, we first focus on what was similar in the first and second phases, and then we present the factors relevant to choosing a coordination strategy (Van de Ven et al. 1976).

The programme organization consisted mainly of the same people at the end of the first phase and the start of the second phase. There were no major changes in the overall goals and aims of the programme, and the programme worked in the same physical office area with physical proximity across the whole programme.

For unit size, the total size of the programme was moderately larger in the second phase. In both phases, we describe the programme as a very large development programme with 10 development teams and a maximum of 130 participants in the part of the programme studied. However, there was a large increase in unit size at the team level, as the teams were now composed of both people from the business side and people from development. The programme had larger team sizes than recommended in agile practices during both the first and second phases. We describe the unit size as *large*.

As for task interdependencies, Van de Ven et al. (1976) defined interdependence as the extent to which unit personnel depended on one another to perform their jobs. They further identified four types of interdependence, from 'independent' work to a 'team'. The transition from a first-generation to a second-generation large-scale agile development method meant that people from the business and development sides who needed to coordinate work on a user story (the requirements in Strode's taxonomy (2016)) were initially in different teams (working in a *sequential* or *reciprocal* mode) but were later placed in the same team. Other types of

knowledge dependencies in which the business side needed technical knowledge were also now managed at the team level. Additionally, what Strode (2016) defined as historical knowledge was now broader when including both business and developers in the same team. Process dependencies were also managed at the team level, and resource dependencies were largely handled at the team level. The restructuring of the programme meant that teams were focusing on a product domain that sought to reduce the number of dependencies on other teams. In practice, however, there were still many dependencies to manage, but the significant difference was that dependencies were, to a much larger degree, handled at the team level. We describe the number of task interdependencies as *high*.

One could argue that task uncertainty was lower in the second phase of the programme, as i) many technical uncertainties were now handled by the platform, ii) the teams were responsible for work within a product domain and iii) programme members had learned both about the domain and technical architecture, as many had worked for over a year in the programme. On the other hand, the programme was i) taking on tasks in an area which was less explored, and ii) all new changes would be implemented on a system which was running and iii) which had grown in size; at the same time, iv) there was more feedback from user groups. Overall, we describe the situation as having a context with high task uncertainty in both phases.

5.2 Coordination Effectiveness

Having described the situations in the phases, we now move our attention to coordination effectiveness. As with developer productivity (Forsgren et al. 2021), we acknowledge that coordination effectiveness is difficult to measure. The tasks in the two phases were very different. One could further expect that there would be a gain in general work productivity as programme participants learned about the domain and the technical system.

Although concluding that the programme was a success is early, many of the benefits described in the business case have started to appear, as described in Section 4.3. Some studies describe project success as a project's capability to deliver on time and within the budget with the expected quality (Ika 2009). The parental benefit programme was completed on time and within the expected budget, and it delivered a solution for which the programme was awarded the annual prize for digitalisation in Norway in 2019.

However, programme success does not necessarily mean that the programme has experienced coordination effectiveness. From our qualitative interviews, we get an impression of perceived challenges and successes in managing dependencies. Edison et al. (2021) listed some challenges identified in prior studies on inter-team coordination, including synchronizing across dynamic and fast-moving teams, addressing meeting overload, decreasing the many handovers between teams as a result of end-to-end development and maintaining transparency across a high number of teams.

In the first phase, we identified 27 coordination mechanisms internally between teams in the development project (CI1) and between the business and development projects (CI2). The informants perceived coordination to work well within the teams and projects, but there were major challenges with coordination between the business and development projects. The development model with teams for phases led to handovers between these two projects. These handovers of solution descriptions of user stories resulted in knowledge dependencies on requirements; the challenge was that there was often a long time from the completion by the business project of a solution description of a user story to the actual development. Clarifying

needs and requirements was frequently time consuming, as people on the business side were fully booked in meetings. As Edison et al. (2021) reported, the number of meetings can threaten coordination effectiveness. Teams experienced synchronization between teams within the projects as working well, but there were indications of a lack of transparency across projects, as the participants in the business project saw the development project as a black box. Some statements suggest that the analysis of needs often resulted in too detailed descriptions, sometimes leading to less autonomy for the developers and sometimes describing work which was not technically feasible.

In the second phase, there was an initial period in which inter-team coordination suffered, as most mechanisms were abandoned, and it was up to the autonomous teams to take the initiative to establish new ones. After an initial phase, however, we identified 14 coordination mechanisms in use. Most informants stated that coordination worked well. They could focus on fewer user stories (reduction of cognitive load) and directly ask people about domain or technical knowledge (manage knowledge dependencies at a lower level); many technical issues were addressed by separate platform teams (also a reduction of cognitive load for team members). One informant appreciated the 'much tighter dialogue'. The change was described as increasing the developers understanding of the domain and the product owner's understanding of the technology, leading to more completed work.

5.3 Coordination Strategies

Given this background on the situation in the first and second phases and the perceived coordination effectiveness, we now discuss the coordination strategies used, which, to a large extent, were derived from the choice of a first- or second-generation agile development method.

The systematic literature review shows that previous studies have identified creating 'dependency awareness' and having 'different arenas for coordination over time' (Edison et al. 2021) as two success factors which particularly relate to coordination. We first describe the coordination strategies in the first phase, followed by the second phase, and then we compare the phases and compare the first- and second-generation large-scale agile development methods. In Section 5.4, we develop five propositions on the impact of transitioning from the first- to second-generation methods.

5.3.1 First Phase

As we show in the results, the first phase relied on a first-generation large-scale agile method, which combined phases and roles and an overall programme organization with central ideas from agile development, such as using scrum at the team level and having an overall flexible product backlog, a team organization, proximity in that the whole programme was co-located and a high presence of the business side through a dedicated project. The coordination mechanisms in the first phase were mainly organized around the phases of development and programme- and team-level roles, and inter-team coordination mainly took place through scheduled meetings. Table 9 shows the characteristics of the two phases.

Most of the coordination mechanisms were stable during the first phase, apart from attempts to remedy the coordination challenges identified between the business and development projects. The new mechanisms introduced included dependency maps (impersonal), and the functional architects were moved physically from the area where the business project was

Table 9 Inter-team coordination in the first and second phases

First phase	Second phase
Phases and projects	Continuous deployment
Explicit documentation: Solution descriptions	Face-to-face communication (proximity)
Roles	Autonomous teams
Scheduled meetings	Unscheduled meetings and face-to-face discussions

located to their development teams to ease informal coordination (unscheduled group meetings and personal horizontal coordination). These measures were not seen as sufficient when starting the last phase of the development project, in which new functionality was to be made in an area that had been less explored, and the programme had to integrate new development with the existing running solution.

Compared with the existing cases in the literature, we can note that there were several more mechanisms for coordination than we found in the study of the enterprise software project reported by Bick et al. (2018). That case illustrated challenges with many unforeseen dependencies, while the first phase in our case experienced challenges with over-specification and the time needed for clarification. Comparing this phase with the Perform programme (Dingsøyr et al. 2018b), we note that both programmes use several coordination mechanisms and organize work in phases and projects. However, the overall organization in the Perform programme made a closer link between the development teams and the projects on architecture, business and test, as most people in these projects worked 50% on a development team. This led to knowledge flow between the four main projects; for example, in the business project, people knew the background of the developers for whom they were writing solution descriptions. In the Parental benefit programme, the business project did not have this knowledge, and some experienced that they wrote solution descriptions which were given to a black box. Comparing coordination in the first phase with what was reported from case studies of the SAFe by Gustavsson (2019), we note that the Parental benefit programme invested much in upfront planning, although it mainly relied on written documentation and not so much on presentations as in product increment planning meetings. Dependency maps were introduced, and there was an overall plan of work until the next release, which corresponded to the board described by Gustavsson (2019).

5.3.2 Second Phase

As described, the change in the second phase to a second-generation large-scale agile development method led to changes in coordination needs. The focus moved from coordination around phases to coordination around the product when transitioning to continuous deployment and autonomous teams. The need for inter-team coordination was reduced, as the management of a number of dependencies was now at the team level. From our data material, it seems that the programme successfully reduced the challenge of knowledge dependencies between the business and development projects by managing these at the team level. The problem with process dependencies in which solution descriptions were finished months before the actual development was also reduced for new user stories, as the whole team was working on the same set of tasks. There were no phases that a user story had to go through, but there was a setup with automatic and manual testing before the daily meeting, in which decisions were taken on the deployment of new functionality (the go/no-go meeting).

Some of the coordination which previously happened in meetings was then coded into the test process. Moving the management of resource dependencies to the team level and making teams responsible for a product area also led to fewer technical dependencies on other teams and fewer challenges with managing resources. Overall, we can say that the second-generation development model led to a transition of coordination work from the inter-team to the team level. However, although the intention was to reduce dependencies between teams as much as possible, inter-team coordination was still needed. The decision to give teams autonomy led to an initial loss of arenas for this purpose. As described in the results, it took several months before a number of coordination mechanisms were re-introduced. With the exception of the go/no-go meeting, techlead forum and change of demo meeting from scheduled to unscheduled (Table 8), the coordination mechanisms in the second phase were similar to the ones in the first phase.

Although we argue that the main changes in coordination strategy involved moving the focus from phases and roles to the continuous deployment of product and autonomous teams, we also note interesting changes in patterns in the inter-team coordination work. The first phase was characterized by many scheduled meetings (11 in total: three arenas for the business project and eight arenas for the development project), as well as the use of unscheduled meetings, the personal mode through one-to-one discussions across teams and the impersonal mode through tools, such as user stories in a wiki and dependency maps. However, in the second phase, we found fewer scheduled meetings (five, including the scrum of scrums meetings). The demo meeting was scheduled in the first phase, but it was changed to an unscheduled meeting in the second phase. We still find personal and impersonal modes for inter-team coordination. In sum, however, we describe the main change as a reduction in scheduled meetings and an increase in informal modes of coordination through unscheduled meetings and face-to-face discussions between individuals (personal, horizontal).

In the determinants of coordination identified by Van de Ven et al. (1976), they found that an increase in unit size led to an increased use of impersonal coordination mechanisms in the greater use of policies, rules and procedures to coordinate activities, as well as to a decrease in the use of scheduled and unscheduled meetings. Our empirical findings show that autonomous teams can manage inter-team dependencies using all mechanisms, but the increase in unit size did not lead to an increased use of the impersonal mode; instead, it led to more mutual adjustment mainly through unscheduled meetings and personal horizontal coordination mechanisms. A possible explanation could be that high task uncertainty and high task interdependence have a greater impact on the coordination strategy. It could also be that the increase in unit size in our case was not sufficiently large to have an impact.

Two propositions on coordination were developed in the study of a large enterprise software programme reported by Bick et al. (2018). First, dependency awareness is necessary but not sufficient for effective coordination. Suppose we accept that the coordination strategy was successful, particularly in the second phase of the Parental Benefit programme. In that case, we note that there was a period in which the programme experienced a lack of coordination after the abandonment of most inter-team coordination mechanisms. With the reintroduction of coordination mechanisms, such as the functional architecture forum, the awareness of dependencies on other teams increased; this, along with other mechanisms, enabled planning alignment, which Bick et al. (2018) proposed as a second proposition for effective coordination. We note that although there were fewer scheduled meetings than in the first phase, there were still arenas for joint planning and review (optional participation in demo meetings), but retrospectives were still at the team level.

Comparing coordination in the second phase to other studies of second-generation large-scale agile development methods, we see that (e.g. compared to Gustavsson's study of SAFe (2019)), the coordination strategy is less dependent on planning meetings, as in the product increment planning in SAFe. Planning was now a continuous process in inter-team coordination meetings between roles, such as the functional architects. Scrum of scrums meetings were reintroduced, but unlike Gustavsson's findings, our informants reported that this arena was working well. The initiation of fora across teams was decided by the teams themselves, much like we see that communities of practice have been initiated and supported by teams at Spotify (Smite et al. 2019) and Ericsson (Paasivaara and Lassenius 2014).

In addition to the major changes in coordination strategy shown in Table 9, we would like to emphasize two points. First, the move to continuous deployment also led to a higher frequency in coordination. The first phase had iterations that lasted three weeks, while the second phase made it possible with much shorter feedback cycles throughout the programme. Continuous deployment was enabled by reorganizing the programme in autonomous teams and the new technical platform, which moved many concerns to platform teams. We did not hear from our informants that they experienced a too heavy workload in coordination activities, and this might have resulted from the autonomy given to the teams, as it was up to them to decide in which arenas to participate. Second, most of our informants saw the second phase as more in line with the principles of agile software development. In his article on sociotechnical coordination, Herbsleb (2007) asked if carefully designed architectures could isolate work at different sites in global software development. In our case, we found that these architectural changes were also important in enabling the autonomy of the teams, which made room for local process differences.

5.3.3 Coordination Mechanisms Over Time

After an initial period, we have shown evidence of increased coordination effectiveness in the second phase, which indicates better congruence between the situation and the coordination strategy. Prior studies (Edison et al. 2021) have indicated that continuous improvement is critical in large-scale agile development, typically organized through team- or programme-level retrospectives. The coordination challenges in the first phase were identified in retrospectives, and actions were taken to reduce the impact of the challenges. Why did the programme wait until the last phase to drastically reorganize? As we described in the case background, there was a strong pressure to deliver after an earlier programme failure. The programme started with a known process and technology to reduce risks. If the programme had changed to a second-generation method earlier or even from the start, it might be that awareness of dependencies would take more time to establish than when relying on many scheduled meetings at the start. The reliance on scheduled meetings can also be seen in other large-scale agile development programmes (Dingsøyr et al. 2018b; Hobbs and Petit 2017).

A critique of large-scale agile development methods is that they provide static advice on coordination (Gustavsson 2019), prescribing a minimum setup with scheduled meetings, an organization relying on teams, regular interactions with stakeholders and, in some cases, specific roles, such as the release train engineer role in SAFe. From prior studies, we have also seen that coordination mechanisms are dynamic structures that change over time (Dingsøyr et al. 2018c). However, there is little advice in second-generation methods on how coordination mechanisms can be tailored to the situation at hand. Our study shows the

impact of change in coordination strategy, although determining which improvements in coordination effectiveness stem from shortening feedback loops is difficult by going from iterations to continuous deployment, by going from focusing on roles and projects to giving teams autonomy or by changing the mode of coordination from mainly relying on scheduled meetings to mainly relying on unscheduled meetings and personal horizontal coordination. Our rich description of the change in coordination strategy shows what Jarzabkowski et al. (2012) described as an absence of coordination, mainly of knowledge dependencies between the business and development projects. Furthermore, efforts to fill this absence with minor changes to the first-generation method were not successful in achieving coordination effectiveness. The coordination challenges were first solved (new coordination mechanisms emerged) when transitioning to the second-generation method in the second phase. However, this change introduced new challenges for inter-team coordination, as old arenas were abandoned. It required time until the new mechanisms stabilized in a situation described by the informants as having high coordination effectiveness.

5.4 Coordination Strategies in the First- and Second-Generation Methods: Five Propositions

Summarizing the changes using van de Ven et al.'s (1976) framework, we see the following:

- A change in the use of the impersonal mode – less written handovers between the business and development functions but the use of impersonal coordination through the technical infrastructure
- More horizontal individual mode – more direct coordination within the teams
- Fewer scheduled meetings – a dramatic reduction in scheduled meetings; it was up to the teams to decide what arenas to use.
- More unscheduled meetings – smaller meetings within and between teams. Other teams' programme participants who were not fully booked in meetings but were available.

After describing the phases and coordination over time, we discuss the central characteristics of the first- and second-generation large-scale agile methods. We develop five propositions (see Table 10) based on our findings and the discussion of prior studies. As described in the background, first- and second-generation methods differ with respect to their main principles and practices. However, as we have seen in the results section, coordination requires significant effort and many arenas, both when using a first- and a second-generation method. This leads to the following:

Proposition 1:

Large-scale agile inter-team coordination requires a combination of group, personal and impersonal modes for the effective management of knowledge, process and resource dependencies.

In line with the findings from other studies of large-scale agile development (Dingsøyr et al. 2018c), we found that scheduled meetings were fundamental coordination mechanisms in the early stages when using a first-generation large-scale agile method. This leads to the following:

Proposition 2:

Scheduled meetings are important in the early phase of large-scale agile development programmes to build knowledge of domain and technical expertise, establish inter-team processes and manage resource dependencies.

Furthermore, when a second-generation method was adopted in the second phase, a new technical platform enabled continuous delivery, which increased the feedback speed. It was mainly an impersonal coordination mode, but it also involved the new (but short) go-no-go-meeting to decide on deployment. This enabled team autonomy, as many dependencies were moved from the inter-team to team levels. Placing both business and development people in cross-functional teams led to fewer handovers, and requirement dependencies, in particular, were managed at a low level.

Proposition 3:

Organizing work around the product instead of projects and phases reduces inter-team coordination needs and thus contributes to the more efficient management of requirement dependencies.

Thus, the second-generation method enables work that is more in line with the key principles of agile development.

However, we observe a lack of coordination after the transition to the second-generation method. Other studies have shown a significant risk of coordination breakdowns if dependencies are not managed at the correct levels. We speculate that if the programme had adopted a second-generation method early, it could risk even more breakdowns.

Proposition 4:

A transition from a first-generation to a second-generation large-scale agile method requires significant domain and technical knowledge amongst programme participants.

Finally, as some old mechanisms were re-established, the programme was perceived to achieve high coordination effectiveness. Many of the roles at the programme level were removed, and supporting functions established in the last phase were also reduced or removed. Overall, the programme was able to move resources from coordination to development.

Proposition 5:

Second-generation large-scale agile development methods, compared with first-generation methods, achieve coordination through the more efficient use of resources.

Summarizing our discussion, we see that large-scale agile development methods will impact the coordination strategy. The determinants suggested by Van de Ven (1976) might need to be supplemented by other factors, such as domain and technical knowledge and experience with agile approaches, when choosing between the first- and second-generation methods. What factors are important for that choice is beyond the scope of our paper. We conclude that choosing first- or second-generation agile development methods will have significant implications on the coordination strategy in that specific mechanisms are given priority and other mechanisms are restricted.

Could there be other explanations for the improved coordination effectiveness? As we have mentioned, one could expect that the whole development organization would become more productive over time, as it learns about the technical product and the domain. However, we have shown that there is a significant change in the use of coordination mechanisms, and if learning should explain improvement, we would have expected to see such an improvement earlier and not after the transition. The informants reported new coordination challenges after the transition, which is also an argument for why the transition impacted the coordination mechanisms.

Is it correct to describe the changes as a change in the whole development method and not improvements caused by autonomy and continuous deployment? We see autonomy and continuous deployment as key characteristics that show a more agile approach than in the first-generation methods. These changes also impacted the number of roles, decision-making authority and the speed of decision making and learning. We think the change was so fundamental that it is correct to describe it as a transition from one generation of methods to the next.

5.5 Limitations and Evaluation

There are several limitations to the chosen approach. We discuss construct, internal and external validity, as well as reliability (Runeson and Höst 2009):

Construct validity To ensure *construct validity*, we have built on established constructs, such as coordination mechanisms, but in the interview guides, we used wording such as 'dependencies' and 'arenas to manage dependencies'. We acknowledge limitations in how we measure constructs in Fig. 1, such as project success and coordination effectiveness. We are formulating theory in a field in which there is no unified agreement on how to measure what is better. As with developer productivity (Forsgren et al. 2021), different groups can perceive coordination effectiveness differently.

Internal Validity We have discussed possible alternative explanations for the changed perceptions of coordination effectiveness and have sought to document the coordination challenges in each phase through multiple sources of evidence (interviews, group interviews, observations, documents). As described in the methods section, we cover a number of roles but

Table 10 Propositions which can form a basis for a new theory on coordination for the particular context of large-scale agile development

Proposition	Description
1	*Large-scale agile inter-team coordination requires a combination of group, personal and impersonal modes for the effective management of knowledge, process and resource dependencies.*
2	*Scheduled meetings are important in the early phase of large-scale agile development programmes to build knowledge of domain and technical expertise, establish inter-team processes and manage resource dependencies.*
3	*Organizing work around the product instead of projects and phases reduces inter-team coordination needs and thus contributes to the more efficient management of requirement dependencies.*
4	*A transition from a first-generation to a second-generation large-scale agile method requires significant domain and technical knowledge amongst programme participants.*
5	*Second-generation large-scale agile development methods, compared with first-generation methods, achieve coordination through the more efficient use of resources.*

not all roles in the programme, and we have not interviewed participants from all teams. We have presented our preliminary findings to the case participants and fellow researchers.

External Validity A typical weakness in building theory from case studies is that a theory can be overly complex or be a 'narrow and idiosyncratic theory' (Eisenhardt 1989, p. 547). Do our propositions have the right scope (Sjøberg et al. 2008)? We have sought to overcome these weaknesses by building on established theory and constructs and by arguing that the propositions are likely to hold for instances of first- and second-generation agile development methods other than the instances in our empirical case. One might argue that large-scale agile development is not a common phenomenon and that the propositions are too narrow, but we believe that it is an important area, showing that methods work with certain adjustments in an area few had thought possible when agile methods were initially formulated.

Has this study generated new insights, is the new theory supported by evidence and have we ruled out rival explanations (Eisenhardt 1989)? We argue that the novel propositions represent a major step forward in our understanding of coordination in large-scale agile development and that we have established new concepts in the form of first- and second-generation large-scale agile development methods that will clarify the differences between approaches. The propositions are supported by evidence from multiple sources, and we have provided a rich description of the context. We have discussed what we see as the main rival explanation.

Reliability A large-scale agile development programme is a complex unit of analysis. We have attempted to cope with this complexity by engaging a large research team (Ribes 2014). A large team meant that we needed to be aligned internally by jointly developing semi-structured interview guides and using a tool for qualitative analysis and a shared file repository as our case database. The method section describes the analysis process and steps in theory development, while the results section shows tracability to data through informant quotes and the narrative.

6 Conclusion

Coordination has been a key concern in large-scale agile software development (Dingsøyr et al. 2019b; Edison et al. 2021). This development is characterized by high uncertainty about how tasks should be solved, a large number of interdependencies between tasks and a high number of people involved—what van de Ven et al. (1976) described as a high unit size.

Coordination has long been a key topic in global software engineering. Herbsleb (2007, p. 9) concluded that for coordination problems, we lack an understanding of tradoffs between tools, practices and methods and an understanding of when the solutions are applicable.

We have reported on two phases of a very large-scale development programme, provided a background of the programme's situation in each phase and discussed coordination efficiency and coordination strategies. We contribute to the discussion on the conditions of the applicability of coordination mechanisms in large-scale agile development and the tradeoffs between coordination mechanisms.

We describe the first phase as first-generation large-scale agile development, combining advice from agile methods with advice from project management. The second phase replaced

advice from project management with current ideas in software development in what we describe as second-generation large-scale software development, with 10 teams that were given significant autonomy and were reorganized into teams after product domain, delivering on a new platform. The change led to a massive increase in the number of deployments of the product from twice a year to daily deployments. We have investigated this change in the focus of coordination from the first focus on the phases of identifying needs, describing a solution, implementing and testing to coordinating around the product. The change was generally perceived as successful, with the programme receiving a prize for digitalisation and our informants appreciating the much tighter dialogue; the latter was characterized as leading to a better understanding of the domain for developers and a better understanding of the technology for product owners.

We have explained the change from the first- to second-generation large-scale agile development methods as having a major impact on coordination. The coordination mechanisms were decided on by the teams themselves when using the second-generation method; there were fewer intermediaries, and the reduction of dependencies between teams led to a decrease in inter-team coordination and an increase in intra-team coordination.

Our findings have implications for theory in that we have established the concepts of first- and second-generation large-scale agile developments, which can make future studies conceptually clearer. Compared with the initial findings on inter-team coordination (Edison et al. 2021), we develop propositions that we hope can form a basis for a new theory on coordination for the particular context of large-scale agile development.

For practitioners, we believe that the main implication of our findings is that they show the implications of change in a coordination strategy. Many organizations are considering large-scale agile methods (Dingsøyr et al. 2019b; Edison et al. 2021), but our study suggests that the choice of a coordination strategy might be a more important question than the selection of a method. We also provide a rich description of changes in coordination mechanisms when transferring from the first- to second-generation large-scale agile development frameworks, which will be helpful in the many organizations currently in this process.

With the extended use of second-generation methods, we hope that future studies could first test the propositions in contexts other than that of our study, with other instances of first- and second-generation large-scale agile methods and other configurations of project complexity, uncertainty and project success. Second, we suggest exploring changes in coordination practices over time in arrangements in which much of the coordination is at the team level—in environments with a high degree of autonomy. Third, we hope that studies on coordination could further our understanding of inter-team coordination by examining coordination between different types of teams, for example, the types suggested in the practitioner literature, such as feature teams and platform teams, as well as other supporting teams in organizations (Skelton and Pais 2019).

Appendix 1: Interview guides

Interteam coordination, third round

- Could you describe the programme?
- Have there been changes in the organization? How/why?
- Could you describe your role in the programme?
- Have you had similar roles previously? What is different in this programme?

- What do you perceive as the main challenges in your role?
- Who do you relate to in your role?
- Have there been any changes in how you work?
- In which circumstances do you need to coordinate with other teams?
- Which dependencies do you have on other teams? Examples?
- How do you manage dependencies?
- Which arenas have you used to manage dependencies?
- How would you describe coordination effectiveness?
- Have there been changes to how dependencies are managed?
- Have there been changes to how the programme is organized? If so, how did you experience changes?
- How do you coordinate with external stakeholders?
- How do you coordinate with external programmes/products/processes in the customer organization? Examples?

Work method, third round

- What work practices do you use
  - internally in the team,
  - across teams,
  - with respect to architecture and
  - against external stakeholders?

- Could you describe a typical iteration at the start of the programme
- Could you describe the last or current iteration?
- Have your ways of working changed over time?
- What has changed?
- Why?
- Who initiated the change?
- The change occurred at what level?

Acknowledgements We are very grateful to all interview participants and contact persons at the Norwegian Labour and Welfare Administration, "Alpha" and Sopra Steria. We thank Kathrine Vestues, who participated in some of the data collection for this article. We also thank Diane Strode from Whitireia Polytechnic (New Zealand), Parastoo Mohaghegi at the Norwegian Labour and Welfare Administration and three anonymous reviewers for their comments, which have helped improve the quality of our manuscript significantly. We are grateful to master's student Camilla Tøftum Ranner from the Norwegian University of Science and Technology (NTNU), who conducted three interviews. Likewise, we are indebted to Marius Mikalsen, Nils Brede Moe, Eva Amdahl Seim and Anniken Solem, researchers in the Agile 2.0 project, who were involved in the discussions on parts of the material in the article. We thank Bjørnar Tessem at the University of Bergen for the discussions on large-scale agile development at project seminars and the wider community participating in the international workshop on a large-scale agile development series.

Finally, we acknowledge the assistance of the late Knut Rolland from the University of Oslo. He participated in all three rounds of interviews and initial analysis work. He is deeply missed.

The data collection and first analysis were done in the competence-building project Agile 2.0, supported by the Research Council of Norway through grant 236759 and by the companies DNV GL, Equinor, Kantega, Kongsberg Defence & Aerospace, Sopra Steria and Sticos. NTNU Concept funded further analysis through the project on organizing digital projects. SimulaMet also supported the analysis and writing of the article through the first author's adjunct position.

Authors' contributions (optional: Please review the submission guidelines from the journal regarding whether statements are mandatory) First and second authors: data collection. All authors: analysis, with the second author doing the initial work and the first author leading the final analysis round. The first author obtained access to the case and handled reporting to the case participants. The third and fourth authors conducted the literature reviews and wrote the initial background section, which the first author expanded and revised.

Funding (information that explains whether and by whom the research was supported) Open access funding provided by NTNU Norwegian University of Science and Technology (incl St. Olavs Hospital - Trondheim University Hospital). See Acknowledgements.

Availability of data and material (data transparency) Not applicable.

Code availability (software application or custom code) Not applicable.

Declarations

Conflicts of interest/competing interests (include appropriate disclosures) None.

Additional declarations for articles in life science journals that report the results of studies involving humans and/or animals Not applicable.

Ethics approval (include appropriate approvals or waivers) Study reported to the Norwegian Centre for Research Data (reference 848,084).

Consent to participate (include appropriate statements) Not applicable.

Consent for publication (include appropriate statements) Not applicable.

References

Ågerfalk PJ, Fitzgerald B (2006) Flexible and distributed software processes: old petunias in new bowls? Commun ACM 49(10):26–34. https://doi.org/10.1145/1164394.1164416

Baham C, Hirschheim R (2021) Issues, challenges, and a proposed theoretical core of agile software development research. *Inf Syst J*, *n/a*(n/a). https://doi.org/10.1111/isj.12336

Bass JM (2019) Future trends in agile at scale: a summary of the 7th international workshop on large-scale agile development. Paper presented at the XP2019, Montreal, Canada

Batra D, Xia W, VanderMeetr D, Dutta K (2010) Balancing agile and structured development approaches to successfully manage large distributed software projects: a case study from the Cuise line industry. Commun Assoc Inf Syst 27(1):379–394

Begel A, Nagappan N, Poile C, Layman L (2009) *Coordination in large-scale software teams.* Paper presented at the proceedings of the 2009 ICSE workshop on cooperative and human aspects on software engineering

Bentley C (2010) PRINCE2 revealed (2nd ed). Elsevier, Oxford, p 296

Bernhardt HB (2022) Digital transformation in NAV IT 2016–2020: key factors for the journey of change *digital transformation in Norwegian enterprises*: springer. pp. 115-134

Berntzen M, Hoda R, Moe NB, Stray V (2022) A taxonomy of inter-team coordination mechanisms in large-scale agile. IEEE Trans Softw Eng:1–1. https://doi.org/10.1109/TSE.2022.3160873
Berntzen M, Stray V, Moe NB (2021) Coordination strategies: managing inter-team coordination challenges in large-scale agile: springer international publishing. pp. 140-156
Bick S, Spohrer K, Hoda R, Scheerer A, Heinzl A (2018) Coordination challenges in large-scale software development: a case study of planning misalignment in hybrid settings. IEEE Trans Softw Eng 44(10):932–950. https://doi.org/10.1109/TSE.2017.2730870
Biesialska K, Franch X, Muntés-Mulero V (2021). Mining dependencies in large-scale agile software development Projects*: A Quantitative Industry Study*. Paper presented at the Evaluation and Assessment in Software Engineering, Trondheim, Norway. https://doi.org/10.1145/3463274.3463323
Bjørnson FO, Wijnmaalen J, Stettina CJ, Dingsøyr T (2018) Inter-team coordination in large-scale agile development: a case study of three enabling mechanisms. Paper presented at the XP2018, Porto, Portugal
Boehm B (2002) Get ready for agile methods, with care. IEEE Computer 35(1):64–69
Boehm B, Turner R (2003) Balancing agility and discipline: a guide for the perplexed. Addison-Wesley, Boston, p 305
Carroll N, Bjørnson FO, Dingsøyr T, Rolland K-H, Conboy K (2020) Operationalizing agile methods: examining coherence in large-scale agile transformations. Paper presented at the international conference on agile software development, Eighth International Workshop on Large-Scale Agile Development
Cataldo M, Herbsleb JD (2012) Coordination breakdowns and their impact on development productivity and software failures. IEEE Trans Softw Eng 39(3):343–360
Dikert K, Paasivaara M, Lassenius C (2016) Challenges and success factors for large-scale agile transformations: a systematic literature review. J Syst Softw 119:87–108. https://doi.org/10.1016/j.jss.2016.06.013
Dingsøyr T, Bjørnson FO, Moe NB, Rolland K, Seim EA (2018a, May 27) Rethinking coordination in large-scale software development, Gothenburg, Sweden
Dingsøyr T, Dybå T, Gjertsen M, Jacobsen AO, Mathisen T-E, Nordfjord JO et al (2019a) Key lessons from tailoring agile methods for large-scale software development. IEEE IT Prof 21(1):34–41. https://doi.org/10.1109/MITP.2018.2876984
Dingsøyr T, Fægri T, Itkonen J (2014) What is large in large-scale? A taxonomy of scale for agile software development. In: Jedlitschka A, Kuvaja P, Kuhrmann M, Männistö T, Münch J, Raatikainen M (eds) Product-Focused Software Process Improvement, Lecture Notes in Computer Science, vol 8892. Springer International Publishing, pp 273–276
Dingsøyr T, Falessi D, Power K (2019b) Agile development at scale: the next frontier. IEEE Softw 36(2):30–38. https://doi.org/10.1109/MS.2018.2884884
Dingsøyr T, Jørgensen M, Carlsen F, Carlström L, Engelsrud J, Hansvold K, . . . Sørensen KO (2022) Enabling autonomous teams and continuous deployment at scale: key lessons from a transition to a more agile delivery model during project execution. IEEE IT Professional
Dingsøyr T, Moe NB (2013) Research challenges in large-scale agile software development. ACM Software Eng Notes 38(5):38–39. https://doi.org/10.1145/2507288.2507322
Dingsøyr T, Moe NB, Fægri TE, Seim EA (2018b) Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. Empir Softw Eng 23(1):490–520. https://doi.org/10.1007/s10664-017-9524-2
Dingsøyr T, Moe NB, Seim EA (2018c) Coordinating knowledge work in multi-team programs: findings from a large-scale agile development program. *Proj Manag J 49*(6):64–77. https://doi.org/10.1177/8756972818798980
Dingsøyr T, Nerur S, Balijepally V, Moe NB (2012) A decade of agile methodologies: towards explaining agile software development. J Syst Softw 85(6):1213–1221. https://doi.org/10.1016/j.jss.2012.02.033
Duncan WR (2017) A guide to the project management body of knowledge, 6th edn. Project Management Institute, Newtown Square
Edison H, Wang X, Conboy K (2021) Comparing methods for large-scale agile software development: a systematic literature review. IEEE Trans Softw Eng:1–1. https://doi.org/10.1109/TSE.2021.3069039
Eisenhardt KM (1989) Building theories from case study research. Acad Manag Rev 14(4):532–550. https://doi.org/10.2307/258557
Espinosa JA, Slaughter SA, Kraut RE, Herbsleb JD (2007) Team knowledge and coordination in geographically distributed software development. J Manag Inf Syst 24(1):135–169
Firth BM, Hollenbeck JR, Miles JE, Ilgen DR, Barnes CM (2015) Same page, different books: extending representational gaps theory to enhance performance in multiteam systems. Acad Manag J 58(3):813–835
Fitzgerald B, Stol K-J (2017) Continuous software engineering: a roadmap and agenda. *J Syst Softw 123*:176–189. https://doi.org/10.1016/j.jss.2015.06.063
Forsgren N, Storey M-A, Maddila C, Zimmermann T, Houck B, Butler J (2021) The SPACE of developer productivity: There's more to it than you think. Queue 19(1):20–48

Gustavsson T (2019) *Dynamics of inter-team coordination routines in large-scale agile software development.* Paper presented at the in proceedings of the 27th European conference on information systems (ECIS), Stockholm and Uppsala, Sweden, June 8-14

Herbsleb JD (2007) *Global software engineering: the future of socio-technical coordination.* Paper presented at the future of software engineering (FOSE'07)

Hobbs B, Petit Y (2017) Agile methods on large projects in large organizations. Proj Manag J 48(3):3–19

Hoda R, Salleh N, Grundy J (2018) The rise and evolution of agile software development. IEEE Softw 35(5):58–63

Ika LA (2009) Project success as a topic in project management journals. Proj Manag J 40(4):6–19

Jarzabkowski PA, Le JK, Feldman MS (2012) Toward a theory of coordinating: creating coordinating mechanisms in practice. Organ Sci 23(4):907–927. https://doi.org/10.1287/orsc.1110.0693

Johnson P, Ekstedt M, Jacobson I (2012) Where's the theory for software engineering? IEEE Softw 29(5):96–96

Kraut RE, Streeter LA (1995) Coordination in software development. Commun ACM 38(3):69–81

Kula E, Greuter E, Van Deursen A, Georgios G (2021) Factors affecting on-time delivery in large-scale agile software development. IEEE Trans Softw Eng

Langley A (1999) Strategies for theorizing from process data. Acad Manag Rev 24(4):691–710

Malone TW, Crowston K (1994) The interdisciplinary study of coordination. ACM Comput Surv (CSUR) 26(1): 87–119

Marks MA, DeChurch LA, Mathieu JE, Panzer FJ, Alonso A (2005) Teamwork in multiteam systems. J Appl Psychol 90(5):964

Mintzberg H (1989) Mintzberg on management: inside our strange world of organizations. Free Press, New York, p 418

Moe NB, Dingsøyr T, Rolland K (2018) To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software development. *Int J Inf Syst Proj Manag 6*(3):45–59. https://doi.org/10.12821/ijispm060303

Mohagheghi P, Lassenius C (2021) *Organizational implications of agile adoption: a case study from the public sector.* Paper presented at the proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering

Nerur S, Mahapatra R, Mangalaraj G (2005) Challenges of migrating to agile methodologies. Commun ACM 48(5):72–78

Okhuysen GA, Bechky BA (2009) Coordination in organizations: an integrative perspective. Acad Manage Ann 3:463–502

Paasivaara M (2017) *Adopting SAFe to scale agile in a globally distributed organization.* Paper presented at the 2017 IEEE 12th international conference on global software engineering (ICGSE)

Paasivaara M, Lassenius C (2014) Communities of practice in a large distributed agile software development organization - case Ericsson. Inf Softw Technol 56(12):1556–1577. https://doi.org/10.1016/j.infsof.2014.06.008

Paasivaara M, Lassenius C, Heikkila VT (2012) Inter-team coordination in large-scale globally distributed scrum: do scrum-of-scrums really work? *Proceedings of the ACM-IEEE international symposium on empirical software engineering and measurement*. New York: IEEE. pp. 235–238

Pries-Heje L, Pries-Heje J (2011) Why scrum works: a case study from an agile distributed project in Denmark and India *AGILE conference (AGILE), 2011*: IEEE. pp. 20-28

Ribes D (2014) *Ethnography of scaling, or, how to a fit a national research infrastructure in the room.* Paper presented at the proceedings of the 17th ACM conference on computer supported cooperative work & social computing

Rolland KH, Fitzgerald B, Dingsøyr T, Stol K-J (2016) Problematizing agile in the large: alternative assumptions for large-scale agile development. In: International Conference on Information Systems, Dublin

Runeson P, Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14:131–164

Sablis A, Smite D, Moe N (2021) Team-external coordination in large-scale software development projects. J Softw: Evol Process 33(3):e2297. https://doi.org/10.1002/smr.2297

Schwaber K, Beedle M (2001) Agile software development with scrum. Prentice Hall, Upper Saddle River

Sharp H, Robinson H (2007) Collaboration and co-ordination in mature eXtreme programming teams. Int J Hum Comput Stud 66:506–518

Sharp H, Robinson H (2010) Three 'C's of agile practice: collaboration, co-ordination and communication. In: Dingsøyr T, Dybå T, Moe NB (Eds., ed) Agile Software Development: Current Research and Future Directions. Springer Verlag, Berlin Heidelberg

Sjøberg DI, Dybå T, Anda BC, Hannay JE (2008). Building theories in software engineering *guide to advanced empirical software engineering*: springer. pp. 312-336

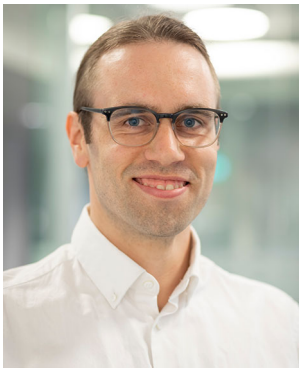
Skelton M, Pais M (2019) Team topologies: organizing business and technology teams for fast flow. It Revolution, Portland, p 216

Šmite D, Moe NB, Ågerfalk P (2010) Agility across time and space: implementing agile methods in global software projects. Springer Verlag, Berlin Heidelberg, p 341
Smite D, Moe NB, Levinta G, Floryan M (2019) Spotify guilds: how to succeed with knowledge sharing in large-scale agile organizations. IEEE Softw 36(2):51–57. https://doi.org/10.1109/MS.2018.2886178
Stol K-J, Goedicke M, Jacobson I (2016) Introduction to the special section—general theories of software engineering: new advances and implications for research. Inf Softw Technol 70:176–180
Stray V (2018) *Planned and unplanned meetings in large-scale projects.* Paper presented at the proceedings of the 19th international conference on agile software development: companion
Stray V, Moe NB (2020) Understanding coordination in global software engineering: a mixed-methods study on the use of meetings and slack. J Syst Softw 170:110717. https://doi.org/10.1016/j.jss.2020.110717
Strode D (2016) A dependency taxonomy for agile software development projects. Inf Syst Front 18(1):23–46
Strode DE, Huff SL, Hope BG, Link S (2012) Coordination in co-located agile software development projects. J Syst Softw 85(6):1222–1238
Uludağ Ö, Putta A, Paasivaara M, Matthes F (2021) Evolution of the agile scaling frameworks. Paper presented at the International Conference on Agile Software Development
Van de Ven AH, Delbecq AL, Koenig R Jr (1976) Determinants of coordination modes within organizations. Am Sociol Rev:322–338
Vestues K (2021) *Using digital platforms to promote value co-creation: a case study of a public sector organization.* (PhD), Norwegian University of Science and Technology
Vestues K, Rolland K (2021) Platformizing the organization through decoupling and recoupling: a longitudinal case study of a government agency. Scand J Inf Syst 33(1):103–129
Vlietland J, van Solingen R, van Vliet H (2016) Aligning codependent scrum teams to enable fast business value delivery: a governance framework and set of intervention actions. *J Syst Softw 113*(supplement C):418–429. https://doi.org/10.1016/j.jss.2015.11.010
Williams L, Cockburn A (2003) Agile software development: it's about feedback and change. IEEE Comput 36: 39–43
Xu P (2009) Coordination in large agile projects. Rev Bus Inf Syst 13(4):29
Yin RK (2018) Case study research and applications: design and methods, 6th edn. Sage, Thousand Oaks
Zaitsev A, Gal U, Tan B (2020) Coordination artifacts in agile software development. Inf Organ 30(2):100288

Torgeir Dingsøyr is professor in software engineering – agile at the Department of Computer Science, Norwegian University of Science and Technology. He is further adjunct chief research scientist at the SimulaMet research laboratory. His research has focused on teamwork and learning in software development, as well as development methods for large software projects and programs. He has published in the software engineering, information systems and project management fields.

Finn Olav Bjørnson is a researcher at SINTEF Ocean. He did his PhD and postdoc at the Department of Computer Science, Norwegian University of Science and Technology. His research has focused on knowledge management in software process improvement, use of agile methods in large scale software projects as well as inter team coordination.

Julian Schrof is a PhD candidate at the Institute for Technical Product Development, University of the German Federal Armed Forces Munich, Munich, Germany working on coordination in agile automotive product development.



Tor Sporsem is a Research Scientist in software engineering at SINTEF, Norway. His areas of expertise include agile software development and digital transformation. He leads and participates in research projects in areas such as continuous software engineering, large scale agile development, and the role of information systems in organizations.