Moe, Nils Brede; Šmite, Darja; Paasivaara, Maria; Lassenius, Casper

Finding the sweet spot for organizational control and team autonomy in large-scale agile software development

Published in:
Empirical Software Engineering

DOI:
10.1007/s10664-021-09967-3

Published: 01/09/2021

Document Version
Publisher's PDF, also known as Version of record

Empirical Software Engineering (2021) 26: 101
https://doi.org/10.1007/s10664-021-09967-3

Finding the sweet spot for organizational control and team autonomy in large-scale agile software development

Nils Brede Moe[1,2] · **Darja Šmite**[1,2] · **Maria Paasivaara**[3,4] · **Casper Lassenius**[3,5]

Accepted: 26 March 2021 / Published online: 14 July 2021

Abstract
Agile methods and the related concepts of employee empowerment, self-management, and autonomy have reached large-scale software organizations and raise questions about commonly adopted principles for authority distribution. However, the optimum mechanism to balance the need for alignment, quality, and process control with the need or willingness of teams to be autonomous remains an unresolved issue. In this paper, we report our findings from a multiple-case study in two large-scale software development organizations in the telecom industry. We analysed the autonomy of the agile teams in the organizations using Hackman's classification of unit authority and found that the teams were partly self-managing. Further, we found that alignment across teams can be achieved top-down by management and bottom-up through membership in communities or through dialogue between the team and management. However, the degree of team autonomy was limited by the need for organizational alignment. Top-down alignment and control were maintained through centralized decision-making for certain areas, the use of supervisory roles, mandatory processes, and checklists. One case employed a bottom-up approach to alignment through the formation of a community composed of all teams, experts, and supporting roles, but excluding managers. This community-based alignment involved teams in decision-making and engaged them in alignment initiatives. We conclude that implementation of such bottom-up structures seems to provide one possible mechanism for balancing organizational control and team autonomy in large-scale software development.

Keywords Large-scale agile · Autonomy · Self-management · Communities of practice · Multiple-case study

Communicated by Robert Feldt and Thomas Zimmermann.

✉ Nils Brede Moe
nilsm@sintef.no

[1] Blekinge Institute of Technology, Karlskrona, Sweden

[2] SINTEF Digital, Trondheim, Norway

[3] Aalto University, Helsinki, Finland

[4] LUT University, Lahti, Finland

[5] Simula Metropolitan Center for Digital Engineering, Oslo, Norway

1 Introduction

Large-scale software development has traditionally been recognized as an initiative best approached using conventional top-down management with a clear governance structure, predefined decision-making mechanisms, and roles with varying levels of responsibility. However, the development of large-scale software systems can never be completely predicted beforehand because the system constantly grows in complexity and often needs to deviate from the original specifications. Further, when large-scale projects become more complex, critical, and urgent, they become more difficult to control (Klakegg et al. 2016). The inherent tension between stability and change in large-scale software projects requires a governance strategy that supports both perspectives. As a consequence, many large software development organizations are moving from a top-down to a more balanced governance approach that combines bottom-up and top-down decision making and increasingly adopts alternative organizational models to strengthen the company's ability to change and avoid the challenges of traditional hierarchies (Olsson Holmström and Bosch 2016). In practice, many larger development projects would then follow agile methods at the team level, combined with a project management framework such as the Project Management Body of Knowledge or PRINCE2 (Dingsøyr et al. 2019).

With the rise in popularity of agile software development methods and associated concepts such as autonomous and empowered teams, software companies have found new ways of distributing authority between those who do the work and those who are in managerial positions (Hoda and Noble 2017; Lee and Edmondson 2017). In agile development, the locus of decision-making moves from the project manager to the software development team, and the decision-making process changes from individual and centralized to shared and decentralized (Barney et al. 2009; Hoda et al. 2012; Paasivaara and Lassenius 2014). However, when many autonomous agile teams together work towards the same goal, a lot of additional coordination and management effort is required (Petersen and Wohlin 2010), which in earlier studies on large-scale agile projects was found to be challenging (Paasivaara et al. 2012). Likewise, studies from management science (Ingvaldsen and Rolfsen 2012) suggest that inter-group coordination is a major challenge when groups are empowered. A variety of mechanisms can be used to coordinate different teams, e.g. Dingsøyr et al. (2018) describe 14 mechanisms for inter-team coordination in a large-scale software project and found that such coordination influences the teams' internal processes and decision making. Further, when implementing a shared decentralized decision-making process in large projects with many teams, alignment problems must be tackled. Previous studies have addressed the challenges associated with the lack of alignment among teams (Bick et al. 2017). If each team works independently, team development processes and technical solutions will ultimately differ and may be highly disconnected from one another. Further, the absence of knowledge sharing between teams can lead to duplicated work, misunderstandings, and integration problems. However, when practices in teams become dependent upon other teams' practices, teams can hardly be described as autonomous (Rolland et al. 2016).

To summarize, balancing the need for alignment and achieving the benefits of a decentralized decision-making structure involving many autonomous teams in large-scale software projects remains challenging. Several important questions remain open: If each team were given the authority and responsibility for organizing their work, how can the organization ensure the overall coordination and success of a large-scale project? Who is responsible for the overall compliance with the project goals? What are effective decision-making structures for cross-team coordination? Finally, because agile methods are emergent

(Boehm and Turner 2005), processes, principles, decision-making, and work structures are rarely completely predetermined. Consequently, the authority of an autonomous team and how such a team organizes the work in a large-scale context will need to be adapted to the project and its context. Therefore, to understand team autonomy in large-scale agile environments, there is a need to study multiple projects in various contexts.

The goal of this paper is to deepen our understanding of how to balance team autonomy and decentralized decision-making with the need for organizational control and alignment in large-scale agile development efforts, i.e., what kind of authority teams should have. In addition, we seek to explore where a sweet spot for team autonomy and organizational control might be. Our research is driven by the following research question:

> RQ: How is organizational authority allocated in large-scale agile software development?

To address our research question, we studied two large-scale agile organizations that are at different stages of their agile adoption in different branches of a large international telecom company, Ericsson. In this paper, we use the definition of large-scale software development by Dikert et al. (2016), i.e. as development done in:

> "software development organizations with 50 or more people or at least six teams. All people do not need to be developers but must belong to the same software development organization developing a common product or project, and thus have a need to collaborate".

The rest of the paper is structured as follows: In Section 2, we present related work. Section 3 describes our research method and empirical background. Results are presented in Section 4 and discussed in Section 5. Section 6 concludes the paper.

2 Background and Related Work

Autonomy and self-management have become the new guiding stars for organizations, as they promise to significantly increase employee motivation and job satisfaction (Langfred 2000), as well as boost creativity, innovation, and productivity (Fenton-O'Creevy 1998; Hoegl and Parboteeah 2006; Mathieu et al. 2008). Further, for knowledge-intensive companies to succeed, organizations must empower individuals to contribute information and ideas at all levels. However, while there is a growing number of self-managing organizations (Lee and Edmondson 2017) and participatory management have clear benefits, one obstacle when implementing such a strategy is unit size (Semler 1989). In this section, we describe the concept of autonomous teams, cross-functional software teams, a model that describes the different levels of autonomy in an organization, and finally autonomy in large-scale set-ups.

2.1 Team Autonomy

Autonomy is a central principle in agile methods. However, the notion of autonomous teams is not new and has been studied and described from various perspectives in the past (Hoda et al. 2012); research in this area has been around since Trist and Bamforth (1951) studied self-regulated coal miners in the 1950s. Such teams were described

from the sociotechnical perspective as having $10-15$ cross-trained members guided by a vision, motivated by peer pressure, and taking on the responsibilities of their former supervisors. Members of such teams should actively seek data and feedback to learn how well they are accomplishing their tasks and take corrective actions at their own initiative to improve their performance. In this work, we rely on the widely used definition of autonomous teams by Guzzo and Dickson (1996) in their review on team performance: *"employees that typically perform highly related or interdependent jobs, who are identified and identifiable as a social unit in an organization, and who are given significant authority and responsibility for many aspects of their work, such as planning, scheduling and assigning tasks to members, and making decisions with economic consequence."*

Autonomous teams have been described from an organizational theory perspective. Morgan (2006), in his book "*Images of Organizations*", describes four principles for enabling such teams:

(a) learning to learn, which emphasises the importance of the team's ability to engage in double-loop learning and drive continuous improvement;
(b) minimum critical specification, in which the management describes only the critical aspects required for teams to function effectively;
(c) requisite variety, which is the need for matching the complexity and diversity of the environment; and
(d) redundancy of function.

The three latter principles suggest that teams should be composed of members with a variety of skills to handle the variety in their external environment and that individuals within the team should, to a high degree, be able to assist and replace each other as required. The advantage of cross-functional teams lies much in their capacity to draw knowledge from multiple sources and conduct multiple activities simultaneously, rather than sequentially, which saves time (Brown and Eisenhardt 1995). A cross-functional software development team often consists of members with different roles, e.g., testers, developers, architects, designers, and analysts. However, knowledge sharing in cross-functional agile teams is challenging due to the complexity of the interactions between different specialists and stakeholders. Moe et al. (2009) found that members of such teams often divide tasks according to their role or responsibility, which leads to a focus on their individual performance and less on assisting one other. Low redundancy and high individual autonomy break the fourth principle suggested by Morgan (2006).

While autonomous teams should have responsibilities for many aspects of their work, there is still a requirement for leadership in such teams. Leadership should be diffused rather than centralized (Morgan 2006), which means that leadership is transferred to the person with the key knowledge, skills, and abilities related to the specific task the team is facing at any given moment (Pearce 2004). While the manager maintains leadership for project management duties, the team members lead and are responsible for decisions when they possess the knowledge that needs to be shared during different phases of the project (Hewitt and Walz 2005). Moe et al (2012) found that the main challenges to shared decision-making in agile environments are related to various conflicts, including the need for short-term progress versus long-term quality, full-time versus part-time members, and new development versus maintenance work. In other words, such teams need a clear direction and shared goals.

2.2 Levels of Autonomy

While autonomy and self-management have been found to impact teams positively, the amount of autonomy that is appropriate under specific conditions remains an important but unresolved issue. Autonomy has a range of implications and is affected by a number of factors, including task interdependency, task complexity, and team design (Mathieu et al. 2008). Cohen and Bailey (1997), for example, reported that the positive effects of autonomy were reduced in short-term teams and were conditioned by task complexity. Additionally, Langfred (2007) suggested that flexibility and adaptability, traditionally regarded as benefits of autonomous teams, may also be associated with dysfunctional outcomes, suggesting that organizations must carefully consider the proper level of autonomy and discretion given to teams. Guzzo and Dickson (1996) described autonomous teams as teams that are given significant authority and responsibility for many aspects of their work. Therefore, we will use the expression "level of autonomy" to refer to the amount of authority held by the team.

In her work on the airline and health industries, Gittell (2006) explored the utility of mechanisms other than hierarchical authority. Gittell observed that in cross-functional work processes (across departments and hierarchies), the companies that performed best had higher levels of relational coordination between roles. However, in these contexts, hierarchical authority could still be relied upon when needed, such as in cases of conflict or disagreement. Based on studies on airlines, Hackman (1986) discussed the relationship between different types of performing units and the amount of authority. Hackman paid special attention to how authority was distributed between those who execute work and those whose responsibilities are mainly managerial. The self-managing performing unit has a certain but limited amount of authority, while the self-designing and self-governing unit has more responsibility (see Fig. 1). In agile software development, the team is accorded full authority to do whatever it decides is necessary to achieve the "goal" (Schwaber and

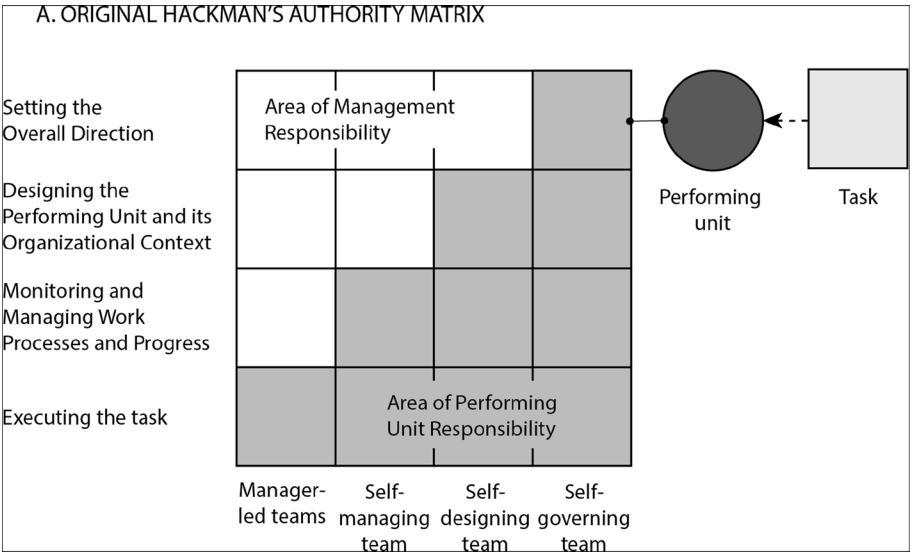

Fig. 1 Hackman's authority matrix (Hackman 1986)

Beedle 2001), however, what this means in a large-scale setting is not understood. Lee and Edmondson (2017), in their study of two large software companies and a large processing company, give examples of how Hackman's understating of authority can be applied. Lee and Edmondson found that authority over work execution was fully decentralized in their cases, but not for other decision domains.

As the motivation for our research is to understand how authority is distributed in a large-scale agile setting with many teams, we use Hackman's work as a basis for our analysis. His work originated from studying teams in a large-scale context, and part of his authority matrix has previously been tested at an organizational level (Lee and Edmondson 2017).

Hackman (1986) explains that a performing unit is a group that consists of several individuals working interdependently on a common task. The performing unit in our study is an agile software team. According to Hackman, in a manager-led team, members have the authority to execute the task only, while a self-managing team is given the responsibility to execute its tasks, as well as monitor and manage its own work processes and progress. As the amount of authority increases, the team is given responsibility for both designing the team and its context and setting the overall direction, and thus teams progress from self-designing to self-governing. A summary of the different levels of team responsibility or authority is shown in Table 1, together with our specific examples. A team may be authorised to perform functions beyond a general level of their given authority or have responsibility for only parts of certain functions.

2.3 Autonomy in Large-Scale Software Development

Research on team performance indicates that the productivity of autonomous work groups is highly situational and that the effects of the practices these groups use depend on such factors as the nature of the workforce and the nature of the organization (Moe et al. 2010). It is, thus, fair to question whether teams in large-scale software development contexts benefit from an increased amount of authority and whether reduced authority brings about unwanted negative impacts on team performance.

Most studies on autonomy are conducted in small-scale one-team projects. Similarly, related theories, such as the one by Hackman (1986), focus on group-level autonomy, where a group is a performing unit working on a task in an organizational context. Even though Hackman's studies were conducted in a large-scale setting (airlines), the interpretation of this theory in the context of large-scale agile software projects is not straightforward. In such projects, many teams are working on tasks that together contribute to the development of a system that is frequently released to customers, which gives constant feedback. Tasks can vary in detail; for example, a task can be a feature or a component. Thus, the system may be viewed as a combination of several independent or interdependent tasks that are to be integrated. Therefore, the level of authority for teams working in large-scale multi-team projects may be limited due to the need to coordinate with other teams and stakeholders (Moe et al. 2019) and the necessity to align the inputs, outputs, and processes across the system development. Further, as teams depend on one other, they often need to wait for other teams to finalize work, which might influence a team's authority. Stray and Moe (2020), in their study of coordination in large-scale distributed agile projects, found access to good coordination and communication tools to be enablers for team autonomy.

Table 1 Types of teams (adopted from Hackman 1986)

Type of team	Level of responsibility	Examples from software projects
Manager-led team	Team members have authority only for executing the task. Managers monitor and manage performance and define the processes, structure the team and its context, and set the overall directions	External team leaders decide who performs which task, how the task shall be solved, and the software development process that the team needs to follow
Self-managing team	Team members have responsibility not only for executing the task but also for monitoring and managing their own performance	The team decides how a new feature should be developed and who should do what. The team monitors the work by seeking data and feedback to learn how well they are accomplishing the development (e.g., speed, quality, and customer satisfaction). The team continuously improves the development process based on the data. The team can help other teams when their own responsibilities are met
Self-designing team	Team members have the authority to modify the design of the team itself **or** aspects of the organizational context in which the team functions. Managers set the direction for these teams but assign full authority to members to do what needs to be done to accomplish the work	In addition to the previous functions, the team can modify itself to solve the work assigned to the team. The team is also responsible for liaising with other teams to solve dependencies (e.g., changes in a sub-system or support from a continuous integration team) and calls for help from others (e.g., UX, architecture, infrastructure, or operations experts)
Self-governing teams	Team members decide what is to be done, structure the team and its context, manage their own performance, and carry out the work	After receiving feedback from the market, the team is responsible for deciding when and if to make a structural course correction to test a new fundamental hypothesis about the product or strategy. Examples of such teams are lean startup teams deciding to pivot or continue

In work systems where teams enjoy high levels of authority, inter-team coordination has been found to be a major challenge (Ingvaldsen and Rolfsen 2012; Lewis and Neher 2007). In large agile projects, there are a number of problems that span development teams as well as decisions that simultaneously affect many teams. Lewis and Neher (2007) found inter-team coordination to be particularly difficult when teams worked independently for different customers, although the applications being built were interdependent. When the team goals conflict with the organizational goals, many teams choose to act in self-interest (Berczuk and Lv 2010), and disrespect the larger context (Farrow and Greene 2008). A study of a large-scale agile environment (Dingsøyr et al. 2018) found that the ability to balance a decentralised decision process and a centralized structure was an important success factor. However, they argue that there is a need to better understand how multi-team development programmes or projects balance inter-team coordination and team authority.

In a literature review of the challenges and success factors of large-scale agile transformations (Dikert et al. 2016), team autonomy was mentioned as both a challenge and a success factor. On one hand, grassroots-level empowerment was highly important for transformation success. On the other hand, team autonomy was challenging to arrange in large-scale settings. In particular, challenges arose if the organization had developed their software development model according to agile principles, and thus allowed teams to operate independently while dependencies existed between the teams, requiring constant collaboration. Further, it was challenging to balance the team goals and the broader goals of the organization. A team could easily place more emphasis on their own goals over the goals of the whole organization. Thus, being part of a large-scale organization might limit the amount of authority that can be given to each team.

Moreover, Dikert et al. (2016) discussed whether the whole organization should conform to one single agile approach or allow the teams to choose their agile approach themselves. A single approach helps the organization adopt a consistent vocabulary, making it easier to collaborate across teams, compare work between teams, and even relocate members more easily to other teams. Giving teams the authority to choose their own approach might be more efficient for the team but might make collaboration across teams more difficult, e.g., if different teams use different agile methods and practices, or even different interpretations of agile. One remarkable example mentioned was that teams with different sprint durations could create problems in delivery. Finally, Dikert et al. (2016) concluded that between projects in the same large-scale organization, different types of agile processes might be required for different types of development, depending on the type of software being developed and the project size. Thus, based on the systematic literature, it is important that in a large-scale agile environment, an organization finds a balance between the amount of authority the team has on the development process and the imperative to conform to the same agile approach as well as to comply with central decisions on dependencies and coordination across teams.

Increasing team authority is a transformational process. Olsson Holmström and Bosch (2016) studied empowerment in seven inherently traditional companies working with large-scale software development, in which selected teams were exposed to increased autonomy and allowed to bypass formal hierarchies. The authors suggest that initiatives that challenge traditional organizational structures still need to align with business goals, existing formal and approved structures, and balance bottom-up decisions (often limited to short-term tactical decisions) with top-down strategies and central decision-making. Importantly, one of

the main reasons for balancing is the need to align the work of multiple teams involved in one programme.

Despite the dearth of research on team autonomy in large-scale agile software development, practitioner reports and agile frameworks touch upon the subject. Perhaps most saliently, the approach used by Spotify (Kniberg 2014a, b; Mankins and Garton 2017) is built around the idea of autonomous agile teams aligned through common goals and a number of bottom-up coordination mechanisms, including communities of practice called guilds (Smite et al. 2020, 2019). The agile coaches at Spotify attempt to balance autonomy alignment across teams (Bäcklander 2019). The major scaling frameworks are more restrictive in their approach to team autonomy. In the Scaled Agile Framework, SAFe (Leffingwell 2018), teams are required to work according to a common high-level rhythm, called the Program Increment, which typically lasts about 12 weeks. The teams can work according to either ScrumXP, an amalgamation of Scrum (Schwaber and Sutherland 2017) and XP (Beck 2000), or team-level Kanban. In the LeSS framework (Larman and Vodde 2016), all teams follow the Scrum process with a synchronized rhythm. In the discussion, we contrast these approaches with our findings in more detail.

3 Research Methods

In this paper, we address the question of how to strike the right balance between team autonomy and organizational alignment in large-scale agile software development. Understanding how much authority to give to teams without jeopardising the overall performance is critical when implementing organizational change to increase the agility and performance of large-scale development. We address this issue by examining ways to allocate organizational authority in large-scale agile software projects, building upon Hackman's authority distribution matrix, and exploring practical ways to make decisions regarding team task execution, managing the ways of working, designing the team, and setting the overall direction. Furthermore, we aim to understand how the various amounts of team authority are perceived by the teams, the reasons for centralizing certain decisions, and the need for cross-team alignment.

In this section, we present our research methods, give an overview of our case projects, and present our data collection and analysis methods.

3.1 Case Study Design

Since the goal of this research is to explore and provide insight into how organizational authority is allocated in large-scale agile software development, it is important to study software development teams in practice. The study reported in this paper is an exploratory multiple-case study (Yin 2009). When conducting the multiple-case study, we followed the five-step process proposed by Yin:

1. Case study design: objectives are defined, and the case study is planned.
2. Preparation for data collection: procedures and protocols for data collection are defined.
3. Collecting evidence: execution of data collection for the studied case.
4. Analysis of collected data.
5. Reporting.

To understand how authority is allocated between the teams and the rest of the organization, we designed a multiple case study of two large-scale agile software development projects at Ericsson, a global leader in telecommunications systems. We selected Ericsson as a subject based on our ongoing long-time collaboration with the company and our knowledge about their existing large-scale development programmes. In our study, we explore the amount of authority agile teams have in comparison to organizational control. Team authority is explored by investigating who is involved in different types of decisions, how much freedom or authority is given to the agile teams, and what decisions are aligned on a project level. As a framework for describing the amount of authority allocated to the teams, we used the four-level authority matrix presented by (Hackman 1986). We structured our data collection according to Hackman's authority areas: 1) setting the overall direction, 2) designing the performing unit and its context, 3) monitoring and managing work processes, and 4) executing the task. We elicited decision points within the four authority areas.

Based on the collected data, we attempted to understand how much organizational control and alignment is needed for self-managing agile teams to be able to collaborate towards a common goal. In contrast to (Olsson Holmström and Bosch 2016), who studied extraordinary teams that, due to special circumstances, are given higher levels of authority, such as teams working in new strategic focus areas and on innovations, or pilot teams adopting new development practices, we study development teams that are working on well-established products. Therefore, we expect that no single team can have full authority in the studied environments. Consequently, we aim to discuss how much authority the teams can and should have and regarding which aspects, as well as how much and what kind of organizational control is needed.

3.2 Study Context

3.2.1 The Company Overview and Case Selection

Ericsson is a leading provider of telecommunication systems and equipment for mobile and fixed network operators. The company develops generic software products offered to an open market and complex compound systems with customer- and market segment-specific versions. Many of these systems consist of many million lines of code and several technologies, and their development and maintenance involve many developers and supporting personnel. Around 2007 – 2009, many divisions of Ericsson committed to an agile transformation, which fundamentally changed their ways of working. For our investigation, we selected two cases that we call Project Mobility and Project Cloud. The projects were selected based on convenience sampling (Patton 2014). Both projects were familiar to us, as we had studied them in relation to other large-scale agile topics and found them interesting and suitable for this study. Project Mobility was part of a research project focusing on determining how to balance the needs for governance and quality control in large-scale distributed organizations and the team's needs for autonomy. Project Cloud was the successor to a project that was deemed exemplary by the organization from the point of view of adopting and using agile methods, and had been used as a benchmarking case for agile adoption internationally within Ericsson. While both selected projects were large and followed an agile software development methodology, the way they implemented agile differed. Project Cloud had, for example, implemented communities of practice (CoPs) (Wenger et al.

2002) as support for scaling agile development, while such parallel structures were not present in project Mobility. Further, the projects came from different divisions within Ericsson and had different backgrounds and histories of their agile transformation. Therefore, by selecting these cases, we expected to find differences in how authority was allocated and how much freedom and authority was given to agile software teams in each of these cases.

3.2.2 Case 1—Project Mobility

Project Mobility, see Fig. 2, is a large-scale development of a software-intensive system responsible for the mobile broadband network. By the time of our investigation, in April and May 2016, the system had evolved over almost 20 years and comprised 30 sub-systems accounting for more than five million lines of code written in different programming languages, and in addition integrated with third-party products. The organization had followed a traditional, plan-driven software development process until 2008, when the agile transformation started and had since implemented many changes to the process and to the governance of the development. Project mobility relied on a Scrum-based (Schwaber and Sutherland 2017) approach.

At the time of our study, the system was developed by around 30 teams from three locations: Sweden, China, and Russia. Our analysis focused on the Swedish part of the project organization. The Swedish teams in Project Mobility were all cross-functional feature teams comprising developers, system managers, testers, a scrum master, and a documentation writer (the latter two were shared among several teams). Although the goal of the company was to enable the teams to do anything anywhere in the system, in practice, teams specialized in the application, platform, or virtualization development, and could also specialize in a number of specific sub-systems.

Due to the large scale, complexity, and number of changes in the organization and the process (e.g., moving from disciplined component teams to cross-functional feature teams), the organization employed rigorous performance monitoring and control. This was performed by a variety of governance roles. Technical coordinators closely review

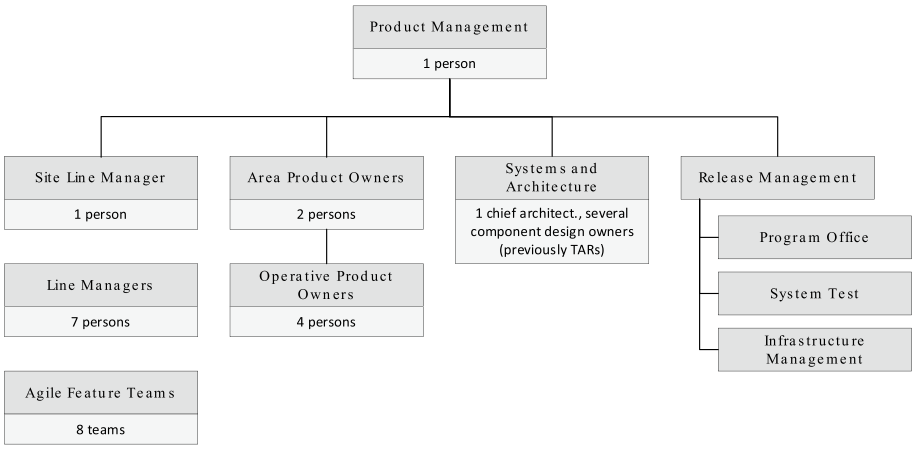

Fig. 2 Organization of Project Mobility at the Swedish site

team progress. Design owners are responsible for the sub-systems, the maintenance of quality throughout the product lifecycle, and the availability of key competence in the area (i.e., key competence should not reside with a single individual). Previously, the technical quality of the team outcomes was reviewed by technical area responsible experts (TARs), who reviewed feature design proposals, code, and final solutions. However, as teams became more experienced, the supervisory functions were minimized to foster team responsibility for their work outcomes, and the TAR role was transformed into a more proactive role called design architect (DA). The DAs bridge product owners, system architects, and teams and act as stakeholders in the early phases of feature development for non-trivial features and features with an architectural impact. Some of these roles require a full-time focus, while others are part-time jobs. For example, the design owner's role is usually assigned to a line manager, while TARs/DAs usually work as senior developers 50% of their time. By the time of our investigation, the teams were expected to be relatively independent, although the supporting organization around the teams was still quite large.

3.2.3 Case 2—Project Cloud

Project Cloud deals with the migration of a core node in the telecommunications network from dedicated hardware to virtualized hardware in the cloud. The original node was, at the time of our study, in use by over 300 telecom operators worldwide. The system is large, consisting of over 30 million lines of code, and its development started over 15 years ago. Until 2009, the system was developed using a waterfall software process. Starting in the spring of 2009, the organization transitioned to a "lean and agile" way of working, in their own words and taking inspiration from the Large-Scale Scrum (LeSS) scaling framework (Larman and Vodde 2016). The main reason for the transformation was the need to shorten the development cycle and build capacity for more rapid reactions to market and customer needs. The development process was Scrum-based, with an iteration length of two weeks.

In 2014, a decision was made to drop the custom hardware and to migrate to a virtualized platform running in the cloud. This migration was organized as a project with separate teams, while other teams continued to develop and maintain the old system in parallel. As the migration project got up to speed and the need for support of the old product was reduced, teams gradually moved from the original system to Project Cloud. During our data collection, around 30 customers were testing the cloud product.

The development organization for Project Cloud consisted of ten teams at two sites located in two countries, Finland and another European country.[1] Here, we focus on the Finnish site, at which eight teams were located, as shown in Fig. 3.

Most teams in Project Cloud were cross-functional, i.e., they were capable of architectural work, development, testing, and integration. The organization aimed for the agile ideal, in which any team would be able to work on any feature, end-to-end. In addition, certain types of specialized testing tasks that could not be done in the feature teams were handled by a couple of separate testing teams.

Cloud migration was technically challenging and represented a greenfield for all project participants, diminishing the knowledge disparity between the technical specialists and the developers. Despite being developed by cross-functional teams, the development

[1] For confidentiality reasons, we cannot identify the second country.

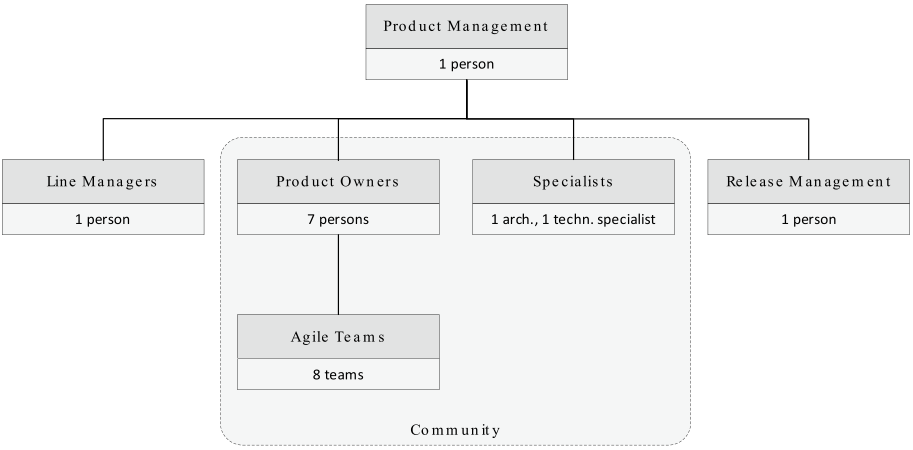

Fig. 3 Organization of Project Cloud at the Finnish site

organization for the old product consisted of several roles outside of the teams. In Project Cloud, on the other hand, as many of these as possible were integrated into the teams in order to build end-to-end capable teams. Only two experts, an architect and a technical specialist studying future technologies, remained outside the teams.

As part of adopting agile methods in the development of the original product, communities of practice[2] (CoPs) were introduced and gained a central role. CoPs were initially used to support the transformation and later, to help coordinate the work between the agile teams (see Paasivaara and Lassenius (2014)).

In the development organization for the old product, several CoPs were formed, including coaching CoPs, Scrum master CoPs, and continuous integration CoPs. Anyone in the organization was welcome to participate in the CoPs, and anyone in the community who wanted to discuss a topic he or she deemed important to the product or the organization could schedule a CoP meeting, invite participants, and start building a community around it. The CoPs were used to discuss topics common to several teams, to coordinate work, and to make product-and development-related decisions. The frequency and regularity of CoP meetings varied. Some CoPs met regularly on an ongoing basis, e.g., the CoPs dealing with coordinating work across teams building a joint feature. Others were arranged ad hoc, e.g., to handle a need for joint architectural planning or joint feature designing.

Using this CoP culture as a basis, Project Cloud further developed it towards what they called *community-based decision-making* (Paasivaara and Lassenius 2019). In this way of working, as explained by our interviewees, the individual CoPs were merged into a single empowered community. All team members and all product owners were members of this community, but managers were not. For decisions with wide impact, the community as a whole makes the decision, rather than a more specialized CoP or a few decision makers. The community had several fixed meeting slots every week, and they were reserved and used as needed.

[2] Wenger et al. (2002) defines a community of practice as *"a group of people who share a concern, a set of problems, or a passion about a topic, and who deepen their knowledge and expertise in this area by interacting on an ongoing basis"*.

3.3 Data Collection

We collected the data through individual and pair interviews. The interviewees were nominated, per our request, by the company representatives involved in our research projects and selected with the goal of reaching maximum variability with respect to the role, experience, and attitudes, as well as to cover the whole development lifecycle. We performed semi-structured interviews with people occupying different roles, both at the team level and in managerial and support positions. We elicited information regarding the interviewee's participation in decision-making and their insights and experiences. Whenever possible, we specifically asked about team authority and non/involvement in decision-making. The interview guides can be found in the Appendix.

In Project Mobility, we had limited opportunities to visit the project site, which was in another city, and therefore decided to conduct pair interviews, as they can generate a large amount of data and include a wide range of responses within a limited timeframe. Another benefit of pair interviews is that participants can build on the responses of the other, which leads to ideas and observations that might not emerge during individual interviews. Kitzinger (1995) notes that the interaction between participants in a group highlights their view of the world, the language they use about various topics, and their values and beliefs about particular situations. Group processes can, according to her, help people discover and elucidate their views in ways that cannot be done in individual interviews. To achieve this effect, our pairs consisted of people with the same role. Pair interviews are, however, not without drawbacks. First of all, the discussions require skilful facilitation from the researcher in terms of allowing the discussion to flow but at the same time keeping the discussion focused on the topic. Another task for the facilitator is to provide space for both participants to contribute. As there were two researchers in the pair interviews, one person led the interview and the other took notes. The one taking notes also supported the main interviewer and made sure all important aspects were covered.

In Project Mobility, we report our findings based on eight interviews with 14 interviewees dedicated to discussing the process of decision-making. These interviews were a part of a larger longitudinal study, in which a total of 28 interviews and seven workshops with teams were conducted, in addition to the analysis of the organizational charts and role descriptions.

In Project Cloud, we conducted four semi-structured interviews. Similar to Project Mobility, these interviews were part of a larger longitudinal study on agile transformation in this organization, during which we performed 78 interviews, see, e.g., (Heikkila et al. 2017; Paasivaara et al. 2013). While comparing the data from Project Mobility and Project Cloud in a meeting among all four researchers, we identified a need for additional data collection in Project Cloud to acquire matching data from both projects on the particulars of decision-making authority. To that end, we conducted four additional interviews with a team member, a product owner, a line manager, and a coach. Thus, in total, we had 82 interviews from Project Cloud. For this paper, the original 78 interviews provided background information on the history, organization structure, processes, and ways of working, as presented in Section 3.2.3. The analysis of decision making and autonomy in the Results section is based on the four new interviews; therefore, we include only these four interviews while presenting the data collection next.

All interviews were recorded and transcribed. The average duration of the twelve main interviews, eight from Project Mobility and four from Project Cloud, included in this study was 73 min. All interviews in Project Mobility were conducted in English,

while in Project Cloud, three of the interviews were conducted in Finnish, and one in English. The quotes used in the paper from Project Cloud have been translated from Finnish by the researchers. A summary of the interviews is available in Table 2. When more than one person attended an interview, this is indicated under the column "interviewee role".

3.4 Data Analysis

The transcribed interviews were imported into a qualitative data analysis software tool, NVivo.[3] We coded using a predefined coding scheme based on the concepts of authority presented by Hackman (1986). According to Hackman, four different functions must be fulfilled when work is done in an organization. We developed Table 3, in which we have provided examples of how we understand these functions in a software development setting. First, someone must set the direction for the team. Second, someone must design the team, arrange for needed organizational support for the work-structuring tasks, decide who will perform them, establish core norms of conduct in the work setting, and make sure people have the resources and support they need. Third, someone must monitor and manage the work process—collecting and interpreting data about how the work is proceeding and initiating corrective action as required. And finally, someone must execute the work. Hackman argued that all these functions need to be present in an organization. Coding for the analysis of data on Project Mobility and Project Cloud was performed by the first two and last two authors of this paper, respectively.

We followed a holistic coding process (Saldana 2009) to understand the four areas of authority discussed in the interviews by assigning pieces of text to categories (''node" in NVivo). We classified the interview fragments related to decision-making in terms of the decision-maker (the team, not the team) or a role (line manager, product owner, etc.). While reading the transcripts, we found different types of decisions that could be aggregated under the four areas of authority. For example, one interviewee discussed his involvement in mandatory courses, which we have added under a new category "Who Decides What Will Improve Practice" under "Monitoring and Managing Work Process and Progress".

Within the open coding, we also coded interesting expressions of opinions, problems, events, good large-scale practices, happenings or actions/interactions, and context-related information.

Creating a common understanding of how to code the four levels of authority (Table 3) was an iterative process. Part of the interviews in Project Mobility was coded first by the two first authors, and then all authors met to discuss the coding process, examples of codes, and how to understand the data. After this meeting, the two first authors completed the coding of Project Mobility, and the two last authors coded Project Cloud. After the coding was completed, we synthesized our findings. First, we revisited the codes related to the four areas of authority and summarized how certain decisions were made in each case project. During the synthesis, we not only highlighted who had the mandate to decide on a certain topic, but also whether the decision was made in consultation with or based on input from others. We had multiple meetings with all authors during the synthesizing process. Illustrative quotations were added to enrich the summaries. An example of the coding process can be found in Fig. 4.

[3] www.qsrinternational.com

Table 2 Data collection through interviews

Project Mobility				
Interviewee role	**Company experience**	**Previous positions at Ericsson**	**Interview duration**	**Time and location**
Development Manager	11 years	Head of R&D, Product Manager, Project Manager, Sales and Marketing Manager	48 min	April and May 2016, Sweden
Two Area Product Owners	16 years	Project Manager, Director of Systems Management	54 min	
	18 years	Program Manager, Project Manager		
Two Operative Product Owners	4 years	–	55 min	
	4 years	–		
Two Scrum Masters	5 years	Software Designer and Developer	56 min	
	18 years	Software Developer, Project Leader		
System Owner	20 years	System Engineer, Line Manager, Network Engineer, System Test Engineer	55 min	
Two Technical Area Responsibles	20 years	Designer, Software Engineer	67 min	
	18 years	Senior Software Engineer		
Two Line Managers/ Design Owners	25 years	Project Manager, Designer	37 min	
	20 years	Head of Department, Business Manager		
Two Line Managers/ Design Owners	6 years	R&D Manager	61 min	
	9 years	Project Manager, Release Leader		
Project Cloud				
Interviewee role	**Company experience**	**Previous positions at Ericsson**	**Interview duration**	**Time and location**
Line Manager, Product Owner	3 years	Product Owner, Developer	59 min	March 2017, Finland
Product Owner	9 years	Developer, Tester	97 min	
Team Member	17 years	Developer, Project Manager, Product Committee Leader	57 min	
Coach	5 years	Scrum Master, Developer	61 min	

Table 3 Examples of how we understand the four functions found in a software development company

Categories	
Setting the overall direction • Who decides what shall be in the system? • Who designs the product architecture? • Who chooses specializations and work items for teams?	Monitoring and managing work processes and progress • Who defines, changes, and improves project processes? • Who defines, changes, and improves team internal processes? • Who monitors team progress? • Who checks quality?
Designing the team and its organizational context • Who decides who is on the team? • Who can remove people from the team? • Who decides which supporting roles are needed for the teams?	Executing team tasks • Who designs features? • Who develops features? • Who delivers features?

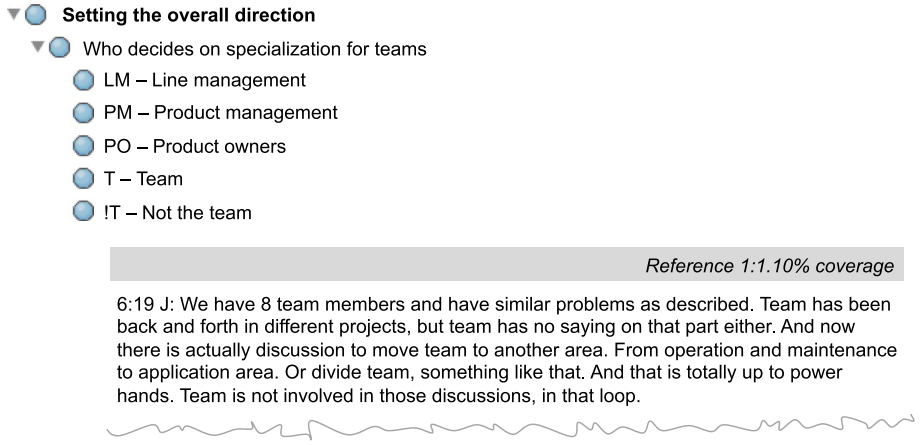

Fig. 4 Example of the coding process

4 Results

According to Hackman (1986), four different functions must be fulfilled when the work is done in an organization. First, someone must set the direction for the organizational work to be done. Second, someone must design the performing unit and arrange for the organizational support necessary for the work. Third, someone must outline, monitor, and manage the work process. Finally, someone must execute the work. We categorized the observed phenomena into these four types of organizational authority (see also Section 2, Fig. 1.).

4.1 Team Authority in Project Mobility

In Project Mobility, we analysed the cross-functional feature teams from Sweden working on feature assignments. Several roles interacted regularly with the cross-functional

teams. These were the operative product owner (usually one per team), the line manager (one per team), the Scrum master (which can be inside or outside the team, or shared among several teams), as well as the design owner and technical area responsibles (TARs) who support the teams during task implementation. The other important roles that emerged in Project Mobility were system owner and the area product owner (one for each product area); the team of architects consisting of a chief architect, software architects, and design architects; and the leadership group, which primarily consists of line managers.

4.1.1 Setting the Overall Direction

The overall direction for the product in Project Mobility was set by the area product owners and the system owner, who had the final decision on what should be in the system. Together, they defined and prioritized the backlog and assigned feature tasks to the teams. The architecture also sets the direction for the product. The product architecture was defined by the team of architects, consisting of experts occupying architecture-related roles from the product and the team level, i.e., design architects who were all part of regular teams and work 50% as senior developers. Teams were expected to consult the design architects whenever the feature they were implementing affected the architecture. In other words, the teams were allowed to change the architecture as long as the changes were approved by the architects.

The features that were assigned to teams, i.e., the specialization of the teams, also affected the direction of the organization and the teams. Specialization was determined by market demand and decided by the area product owners. The lack of involvement in what a team was going to work on was regarded as a limitation and something that reduced a team's level of empowerment, as a Scrum master complained:

> "The team has been back and forth on different projects, but the team has no say on that part either. And now there is actually a discussion to move the team to another area, from operation and maintenance to the application area, or divide the team... And that is totally up to the "power hands". The team is not involved in those discussions, in that loop… We don't know from one day to another day what the day will bring. That is bad for morale".

While the area product owner and the system owner defined which features the teams would work on, teams could contribute to the system vision and content during hackathons, in which the teams were expected to come up with their own ideas beyond those in the backlog. Hackathons are product-specific monthly initiatives that were introduced by the leadership team to provide teams with the platform to influence the direction of the product and elicit innovative ideas. As the development manager described:

> "I also have decided that one day a month every section should conduct innovation day. And they can do whatever innovation or improvement they like. It should eventually lead to better results for Ericsson, but it should not be backlog item APOs order."

Hackathons were quite new for the organization, and not every team or team member had participated in hackathons, and therefore their effect was yet to be determined, as described by a Scrum master:

"Yeah (I participate), we had one last week for section actually. I really like the idea, but it is hard to get time aside. But now this is the thing from above, you should do this one day in a month. Now we try to attend it more and plan for it. I will try. But for the first one I was quite stressed because I was trying to get something working at the end of the day".

4.1.2 Designing the Team and its Organizational Context

The decisions about who belonged to a cross-functional team in Project Mobility were made by the leadership group. Most of these decisions were made without consulting the teams. A line manager explained that while decisions regarding the team size and profile might be discussed with teams, teams were not consulted about certain staffing decisions, especially regarding individuals returning from parental leave, when teams changed specialization, existing teams merged, or new teams created. This, however, was reported to affect team morale, as a Scrum Master explained:

"We got some new members. Not that we should, we just got them. So that is very un-empowering. But great guys and we like them. But we didn't have anything to say about that decision. That was really bad."

The supporting roles for teams were changed continuously by management. For example, the role of TARs was introduced to ensure the quality of team outcomes by focusing on reviewing design proposals, code, and feature solutions. This role was later replaced by the design owner role to give teams more responsibility by focusing on on-demand consultation regarding product architecture in the early phases of development. The introduction, redefinition, and removal of these roles were part of a large improvement process that was run by the leadership group. As in the team staffing decisions, teams were hardly involved.

4.1.3 Monitoring and Managing Work Processes and Progress

The teams in Project Mobility were entitled to a large degree of autonomy regarding their internal ways of working. Further, team retrospectives were used to discuss whether adjustments were needed to find better ways of working and become more productive. A Scrum master described the freedom of choosing how to work as follows:

"We started with Scrum in theory maybe, but we adapted our own ways of working that fit the team. And we are constantly changing things…. we can use Scrum, we can use Kanban for a while. So, there is no overall decision on what to follow."

Team performance was monitored and discussed by each cross-functional team and its close collaborators, i.e., the team members, the operative product owner, and the line manager. Performance monitoring included feedback on the deliverables and their quality. We found that operative product owners and TARs previously handled quality control through mandatory code reviews and demos. However, as the teams became more experienced, the need for "policing functions" was discussed and questioned, as an operative product owner described:

"I see one risk with the TAR role as a gatekeeping role: a team might use it as a safety buffer. If teams think 'Okay, if something is wrong, that will be dis-

> covered by TAR', and I think, we have to get away from that. So, teams take that responsibility, and not rely on someone else to look and make sure that it is really good."

While a team had freedom to choose the work process, there was a high need for aligning the teams; many mandatory processes and even improvements were defined by the line managers and the leadership group. These included a quality checklist before delivery, sprint length, mandatory reporting meetings, and training courses. The inability to influence the mandatory processes, however, was not always well received by the teams, since teams at different levels of maturity were treated equally. Consequently, targeted improvements were sometimes perceived as meaningless if the mandatory solutions addressed a problem that a particular team did not experience, as explained by a Scrum master:

> "Now, what I have not seen before is a lot of directives. Every team should do that and that, instead of work with teams and go forward to find what are the problems with your team, what can you do to improve a team...
> Because someone did that, and that was good, so everyone should do it... instead of trying to focus on what could be improved in different areas in different teams. That I would prefer."

Each team was responsible for unit testing, while integration and system testing were performed by the continuous integration (CI) team.

4.1.4 Executing the Team Task

Teams received tasks from the backlog with features prioritized by the area product owner. Tasks usually contained a brief description, which the team subsequently detailed and designed. Teams interacted with operative product owners when designing the features and could consult design owners if a technical expert opinion or architectural expertise and advice were needed.

When the feature work was completed and tested, a team packaged the feature, verified its completeness by filling in the readiness checklist, and sent it to the continuous integration unit. The final delivery to the customer was undertaken by the release managers.

4.2 Team authority in project cloud

In Project Cloud, all teams were members of the *community*, which was the central decision-making body in the project. This gave the teams authority both on the team and the project level. As stated previously, the community was formed by all project teams and product owners. Under special circumstances, product managers and line managers could take part in the community meetings, but they were not considered community members.

The community made decisions that used to belong to management, including strategic technical issues, as well as process-related ones, e.g., setting the overall development direction and specifying the ways of working. This change of decision-making power from management to the community was a major cultural change in the organization.

4.2.1 Setting the Overall Direction

Given that the product was in a mature market segment, product management retained the main responsibility for the overall direction of the system. Compared to the old project, the technological change from dedicated hardware to a virtualized cloud-based solution provided opportunities for novel features. This, in turn, resulted in the development organization providing both feedback on existing requirements and ideas for completely new ones. The interaction between development and product management changed from a command-and-control style towards one based on trust and dialogue—and a little sense of rebellion against the traditional way of working, as described by a product owner:

> "In practice we don't have top-down management anymore, so if the PO decides something, or if the architect decides something, nobody takes it seriously unless it has the community behind it. The decision has to be made by the community and be accepted by it."
> "The overall direction is set by the community… But it is very challenging, e.g., if they [headquarters] have an architect who has the power to decide, and he comes here and commands us to do something, and then he is told that he cannot decide… that you don't have the power, you can say whatever you want, but the community will decide and make the implementation anyway. So, we have kind of the reputation of a pirate ship in the sense that we have bravely made decisions outside the mainstream that in retrospect have turned out to be very good."

The change from an executing organization towards one intimately involved in deciding what to do was based on the fact that the community in Project Cloud gained valuable knowledge about the possibilities and limitations of the new technology. This gave them the confidence to weigh in on requirements-related decisions and negotiate with product management. A product manager argued that this open discussion and negotiation based on a deep understanding of the product and customers in the community led to better requirements:

> "We [the community] are really a strange creature in the sense that if there comes a requirement [that we think does not make sense], we … tell that it really does not work this way, we won't do it. And in particular, if this requirement comes from a person who has not done hands-on work with the cloud, and he argues against twenty guys [who are technical experts], doing real development work, you can guess whose arguments are better… Or sometimes we question the customer value [the requirement] provides, and that often leads to interesting discussions. At least with our product management. If we tell them that this does not produce customer value, and we can motivate it well, then the requirement is rather quickly forgotten."

In addition to discussing the requirements, the community was responsible for architectural decisions. While the old organization had separate architect roles, in Project Cloud, the technical leads and experts were part of the cross-functional teams. Thus, they were immersed in the technology while participating in the daily technical work rather than living in an "architectural ivory tower" far away from the actual development. This allowed the teams to undertake end-to-end development, as explained by a product owner:

> "We [in the CoP] share information and solve problems together and make decisions. And everybody is on the same line, and the decisions mostly come from the teams

> and the community. In the old days,…, we had a systems department that would study the issue several years beforehand, and would write specifications that were carved in stone and handed over to the developers. Now it is kind of the opposite. Basically, the biggest change was that the old systems experts and system testers were all integrated into the teams…. The teams really became end-to-end capable."

The project had one remaining architect role, although that role had morphed from decision-maker to assistant, supporting the community and helping to ensure that the architecturally significant decisions made by the community conformed to corporate guidelines.

Through the community, the teams felt that they were able to affect the overall direction in the organization, and that the decisions made by the community made use of the best knowledge in the organization. A team member explained how much they could influence the overall direction:

> "In my opinion, quite a lot. In particular, now that we started this virtualized version of our product, there were a lot of people in the community assessing options and pushing it in the right direction. And that is maybe one strength of Ericsson Finland, that we have made the right decisions because we have discussed issues broadly and tried to elicit all knowledge that is available in-house."

While the organization strived to form teams that would be able to work on any feature, without any areas of specialization, this turned out to be tricky in practice. The main concern was related to productivity, and this meant that the product owners mostly decided what each team would work on next. A team member explained:

> "I think it is maybe a bit bad. Of course, they [product owners] would certainly listen to us if we told them that we would like to make more of this and less of this, but they really don't ask us a lot... And if some team has got… kind of stamp… that it focuses on certain types of tasks, the POs, because they focus on throughput, always give the same kind of tasks to that team, which might not be optimal from the point of view of competence development. So, I think that the PO and line management should think a bit about the competence development needs and wants of people."

4.2.2 Designing the Team and its Organizational Context

Although line management was responsible for recruiting and staffing decisions, teams were actively consulted, and their recommendations were typically followed. Many decisions were made based on a dialogue between the teams and the line managers. During a recent external recruitment process, selected team members participated in the job interviews and actively contributed in the technical parts of the interviews, giving them influence on who should be part of the team.

The level of authority given to the teams when it comes to designing the teams and their organizational context can be illustrated by the way a recent major organizational change was handled. Since some teams were too small, half of the teams were designated for restructuring. This was achieved by offering the teams a choice: either the teams drive these decisions, or the line managers drive them in consultation with the teams. A coach described:

> "... The teams thought that it would be better if the line management decided, and teams were only consulted.... It would be fair to say that the line management was a bit frustrated at the slowness of having to discuss with people, hear them out, and then discuss more with somebody else… But the end result, based upon feedback

from the teams, seems to be quite good because the teams themselves feel that they have been able to influence things, and that makes them feel much more comfortable [with the decisions]."

In terms of organizational context, product owners or other members of the community assisted in connecting the teams with the right experts whenever external expertise was needed. As a result, teams in Project Cloud were in direct contact with the other teams and with supporting roles inside and outside the project.

4.2.3 Monitoring and Managing Work Processes and Progress

The decisions related to ways of working and what tools to use were made both at the team and the community levels. If a decision was not in conflict with what had been agreed upon at the community level, nor created problems for other teams, a single team could make the decision. The teams were authorised to define their own ways of working, as long as other teams were not affected. This guideline in Project Cloud aimed to maximize team autonomy, restricting it only to avoid conflict or chaos in the organization. If a team wanted to make a change affecting others or change an agreed-upon community decision, they had to call a community meeting. This was explained to us by a team member:

"The team decides [its working processes] independently... Yes, and for example when some new people came [into the team] around a year and a half ago, we had this session [in our team], where we discussed how to work, how do we want to work, and we created some kind of instructions…. When something has been decided in the community, a team cannot change it, but has to take it back to the community [and suggest changing it]. For example, commonly agreed things, like we use Gerrit [a tool] etc., we cannot decide ourselves that we don't like it and won't use it."

To better understand the type of process-related issues that a team could address independently and what belonged to the community, we asked whether a team could, for example, decide for themselves to use Scrum or Kanban. We learned that since certain alignment points existed between the teams, decisions affecting this alignment required everyone to change, and thus must be made in the community. At the time of the interviews, all teams had synchronized two-week sprints. However, inside the sprints, each team could work the way they wanted. For example, many teams used Kanban boards, some had physical boards, and some had electronic boards.

Each team agreed with their product owner on how team progress and performance were monitored. Teams were required to trace their progress using Kanban or Scrum boards and to hold daily stand-up meetings in which their product owner also participated. The teams considered the daily stand-up meetings as reporting meetings and not team internal meetings held for coordination purposes because of the presence of the product owners. The POs required each team to maintain some kind of board, as well as hold daily stand-ups.

Quality was mainly ensured through automated testing, which was performed on several levels and in different cycles. When working on new features, teams implemented new automated tests as a part of the feature development. Teams were also responsible for continuous integration and the automated testing system, called the "washing machines". Teams took two-week turns in monitoring the washing machines. Therefore, we can say that the teams and the community as a whole were responsible for quality and also for checking quality.

4.2.4 Executing the Team Task

When teams were assigned features to develop by the product owners, they performed the development work independently, taking full responsibility for designing, implementing, and testing the features. In cases where their work affected other teams or the product design or architecture, related decisions were deferred to the community. In the community meetings, other teams and experts, such as architects, weighed in on the decision.

The final delivery to the customer was undertaken by the release managers in collaboration with the product owners, because the preparation of a release requires "bureaucratic" work that the teams were shielded from.

4.3 Level of Team Autonomy in the Case Projects

In Table 4, we summarize the distribution of authority in the two studied projects. The white cells represent the areas and decisions that individual teams are responsible for. The light grey cells represent the areas aligned across the teams and were influenced by the teams or decided by the teams jointly as a community, while the dark grey cells represent areas and decisions made by line management, product owners, or other managerial roles without prior consultations with the teams. Note that product owners are not considered part of the teams, as this was the setup in both our cases. The reason for this was that the POs in Project Cloud, and the OPOs in Project Mobility served several teams. This means that our teams refer to the Development Team in the Scrum guide sense, not the Scrum Team, which includes the PO (Schwaber and Sutherland 2017). The newest version of the Scrum guide (Schwaber and Sutherland 2020) has dropped the concept of Development Team, referring only to Developers instead.

Our intention was to understand the amount of authority given to a single team working in a large-scale software development project, what decisions a single team could influence, and which decisions were made without consulting the teams. Next, we briefly compare the cases.

4.3.1 Setting the Overall Direction

In Project Mobility, the area product owners and the system owner decided what should be in the system, while teams could contribute to the system vision and content by introducing ideas during hackathons. In Project Cloud, having an open discussion with the community had a strong influence on the overall direction by providing both feedback on existing requirements and ideas for new ones, while product management retained the final responsibility for the overall direction of the system.

Further, in Project Mobility, the product architecture was designed by the team of architects that consisted of design architects from the teams, while in Project Cloud the community was responsible for architectural decisions. Finally, in Project Mobility, the team specialization and features developed were decided by the area product owners, while in Project Cloud the product owners decided on team specializations but listened to input from teams regarding what they wanted to work on.

4.3.2 Designing the Team and its Organizational Context

The leadership group in Project Mobility decided who was part of each team and which supporting roles there were, mostly without consulting the teams, which negatively affected team morale.

In Project Cloud, line management took the final responsibility for recruiting new team members, as well as for moving people between the teams, while actively involving the teams in the process. The community assisted teams in connecting to key experts and to the other teams.

Table 4 Summary of the distribution of authority. The white cells represent areas of team responsibility, while the light grey cells represent areas, in which teams can influence decisions or in which teams are consulted. The dark grey cells represent areas in which decisions are made without consulting the teams

Areas of Authority	Types of Decisions	Project Mobility*	Project Cloud*
Setting overall direction	Who decides on team tasks and specialization?	APOs	POs
	Who designs the product architecture?	Team of architects	Community
	Who decides what shall be in the system?	POs and SO, input from hackathons	PMs and POs with the community
Designing the team and its organizational context	Who decides who is on the team?	LM	LM in consultation with the team
	Who is able to remove people from the team?	LM	LM in consultation with the team
	Who decides which supporting roles are needed for the teams?	LM	Team
Monitoring and managing ways of working and progress	Who defines, changes, and improves project processes?	LG	Community
	Who monitors team progress?	Team, PO and LM	Team and PO
	Who defines, changes, and improves team internal processes?	Team or Team, PO and LM	Team or team and the community
	Who checks quality?	Team, PO and LM	Team and the community
Executing the team's tasks	Who designs features?	Team and OPOs	Team
	Who develops features?	Team	Team
	Who delivers the features?	RM	RM and PO

* *PO* product owner, *OPO* operative product owner, *APO* area product owner, *LG* leadership group, *LM* line management, *SO* system owner, *RM* release manager, *PM* product management

4.3.3 Monitoring and Managing Work Processes and Progress

Each team had the main authority to decide their internal process in Project Mobility. However, to align between the teams, many mandatory processes, such as a quality checklist before delivery and sprint length, were defined by the line managers and the leadership group.

Team progress and quality were monitored by the team and their OPO and line manager. Integration and system tests were performed by a separate CI team. In Project Cloud, the overall project processes and changes to it were aligned in the community. The team's internal process and ways of working were decided by each team, as long as they were not in conflict with the community-level agreements. Team progress was monitored by the team and their PO, and teams and the community were responsible for the quality of their work.

4.3.4 Executing the Team Task

In Project Mobility, teams took full responsibility for designing, implementing, and unit testing the features, and were supported by their area and operative product owners, as well as by design owners. In the same way, in Project Cloud, teams took full responsibility for designing, implementing, and testing the features, and were supported by their product owners.

In both projects, the release managers took care of the final delivery to customers.

4.3.5 Comparing the Cases

The results of the cross-case analysis indicate that teams in Project Mobility had similar levels of authority as in Project Cloud when it came to executing team tasks and deciding on team internal processes. However, project-level decisions in Project Mobility were handled to a large extent without consulting the teams, while the teams in Project Cloud either jointly decided or at least influenced decisions with respect to managing ways of working within the project, designing the teams and project context, and setting the overall direction for the system they developed. This was done through a community formed by the teams that included the technical and product experts, as well as the Product Owners. Due to the high levels of technical uncertainty at the beginning of the project, there was a need to combine the competencies and together find the best possible solutions to the challenges that arose. Due to this technical uncertainty, everybody was at the same starting line regarding competencies. The success of the early decisions proved that community-based decisions had a higher likelihood of success than individual decisions, which was the reason to enforce the community as the central decision-making forum, especially regarding technical decisions. The only decisions that were made without consulting the teams for Project Cloud were related to how tasks and specializations were distributed between different teams. Here, teams did not traditionally have much influence on the decisions, as the product owners made the decisions. However, the interviewed team members viewed this as problematic and hoped that teams would also have more influence in this area in the future.

5 Discussion

Large and complex software systems can be characterized by unprecedented scale in terms of lines of code, the amount of data stored, accessed, manipulated, and refined, as well as the number of connections, interdependencies, hardware, computational elements, and customers. Such systems are developed in complex multi-team contexts, where a team needs to network with experts and to align many decisions regarding the tasks and the process with other teams and the organization (Šmite et al. 2017). As a result, team autonomy is reduced. For these reasons, large-scale software projects often have a complex governance structure with centralized authority (Dingsøyr et al. 2018; Olsson Holmström and Bosch 2016). However, agile software development relies on the idea of autonomous teams with broad authority to decide on their own work.

In this paper, we looked at how to balance team autonomy with the need for organizational control and alignment in large-scale agile development efforts. To this end, we examined *how organizational authority was allocated in large-scale agile software development* in two empirical cases at Ericsson—one at a Finnish site and one at a Swedish site. This question is interesting because in agile environments, the focus of decision-making is expected to move from the project manager to the software development team, and the decision-making process changes from individual and centralized to shared and decentralized (Barney et al. 2009). At the same time, the organization needs to ensure that all the agile teams in the large-scale project are working towards the same goal and that work is coordinated between the teams, which is traditionally achieved by formal organizational control. In the following, we discuss how organizational authority was allocated in the studied projects and what we can learn from it with respect to balancing team autonomy and organizational control.

5.1 Authority in Large-Scale Agile Projects

To understand how much authority the teams have, we used the framework proposed by Hackman (1986) and Table 1 to map the distribution of authority in the studied projects (see the results of the mapping in Table 5).

In both case projects, the teams had full control only of executing the tasks. For the other three levels, authority either rested with the management, was shared between the teams and management and/or supporting roles, or was taken in a collective forum. The latter form of decision-making was of particular interest because Hackman (1986) considered that authority either rests with the work unit or with the management (which can consult or allow influence from the work units). Our findings indicate that authority could also formally rest with a development community (shared between teams).

Furthermore, we learned that team authority tended to increase over time. First, in Project Cloud, the introduction of communities reduced management authority since many decisions (e.g., regarding features, process, product, and teams) were now transferred to the teams. Second, in Project Mobility, the level of monitoring functions (external experts checking the work of the team) was reduced, and teams became responsible for quality and compliance management. Our results are in accordance with Langfred (2007), who suggests that organizations must carefully consider the proper level of autonomy and discretion given to teams. While we found evidence of teams claiming to have too little authority, we did not find evidence of teams receiving too much.

Table 5 Authority in our case projects vs. popular agile scaling frameworks. The white cells represent areas of team responsibility, while the light grey cells represent areas, in which teams can influence decisions or in which teams are consulted. The dark grey cells represent areas in which decisions are made without consulting the teams. N/A means that the model does not say anything about it

Areas of Authority	Types of Decisions	Project Mobility*	Project Cloud*	SAFe	LeSS	Spotify model
Setting overall direction	Who decides on team tasks and specialization?	APOs	POs	POs, community in PI planning[a]	PO, APOs and teams in sprint planning one	LG and community
	Who designs the product architecture?	Team of architects	Community	Architects	Teams jointly	LG and community
	Who decides what shall be in the system?	POs and SO, input from hackathons	PMs and POs with the community	PM, POs	PO and APOs	LG, community and team
Designing the team and its organizational context	Who decides who is on the team?	LM	LM in consult. with the team	Development manager in consult. with the team	~~N/A~~	LM in consult. with the team
	Who is able to remove people from the team?	LM	LM in consult. with the team	Development manager	~~N/A~~	LM in consult. with the team
	Who decides which supporting roles are needed for the teams?	LM	Team	Management	Team	LG, community and team
Monitoring and managing ways of working and progress	Who defines, changes, and improves project processes?	LG	Community	Lean and agile competence center and team	Teams, supported by managers	Community and team
	Who monitors team progress?	Team, PO and LM	Team and PO	Team, RTE, POs	Team, APOs and PO	Team
	Who defines, changes, and improves team internal processes?	Team or Team, PO and LM	Team or Team and the community	Team and ART	Team or Teams jointly	Team
	Who checks quality?	Team, PO and LM	Team and the community	Team and ART	Team and Teams jointly	Team
Executing the team's tasks	Who designs features?	Team and OPOs	Team	Team	Team	Team
	Who develops features?	Team	Team	Team	Team	Team
	Who delivers the features?	RM	RM and PO	ART	PO	RM

PO product owner, *OPO* operative product owner, *APO* area product owner, *L* leadership group, *LM* line management, *SO* system owner, *RM* release manager, *PM* product management, *ART* agile release train (Leffingwell 2018)

[a]Product increment planning with all teams in an agile release train is typically held every 12 weeks

When classifying teams in Project Mobility and Project Cloud according to Hackman's authority matrix (Fig. 1), it is fair to say that teams in both large-scale projects have more authority than a typical manager-led unit, but the level of authority is still insufficient to qualify their categorization as self-managed, self-designing, or self-directing units. According to Hackman (1986), self-management occurs when a unit controls the decisions regarding the execution of tasks and the management and monitoring of work and progress. Thus, we consider teams in both projects as partly self-managing, since the authority is often shared with management and/or the community. Further, the influence that individual teams have on higher-level decisions varies. In Project Mobility, the team's authority over the organizational context and the unit's design, as well as the overall direction, is lower than in Project Cloud, in which teams are engaged in community-made decisions.

Due to the popularity of various commercial and non-commercial prescriptive scaling approaches and frameworks, we added three of the most popular ones to our comparison to see how their recommendations match our results and how much autonomy they prescribe. The result is visible in Table 5, where we added the SAFe—Scaled Agile framework (Leffingwell 2018), the LeSS—Large-Scale Scrum framework (Larman and Vodde 2016), and the Scrum adaptation made popular by Spotify, often referred to as the "Spotify Model" (Kniberg 2014a, b). The community in the Spotify Model refers to the guilds (Smite et al. 2020) and other cross-company units with team representatives. The entries in the table are based on our interpretation of the frameworks' prescriptive recommendations and not single cases implementing them.

Looking at team autonomy in the scaling frameworks, SAFe represents the most traditional model, with the highest level of standardization and management responsibility and subsequently the lowest degree of team autonomy. Teams are focused on delivering features in synchronized sprints as part of 12-week release trains, using either Scrum or Kanban processes. Even though Project Mobility did not rely on the SAFe model, the level of team autonomy was close to what the SAFe model prescribes.

The LeSS model has a different take on management, stressing the need for managers to facilitate the development teams, and taking responsibility for systemic improvement of the development organization, but letting the teams and product owners take care of all product decisions, as well as process improvements (Larman and Vodde 2016). This view is visible in allowing teams a higher degree of autonomy, as indicated in Table 5. As mentioned previously, the development model used in Project Cloud was inspired by LeSS, which is also seen in the very similar approach to team autonomy, with the major difference being that Project Cloud explicitly defined a community as a decision-making authority.

The Spotify model, also based on Scrum, has a high level of team autonomy, as the teams are involved in or are able to influence all kinds of decisions, from executing the task to setting the overall direction. Of special importance, as the community in Project Cloud, and a community of teams in the LeSS model, there are also communities of practice in the Spotify model, called guilds. Management seems to have a slightly higher involvement in decision making in the Spotify model than in Project Cloud or in the LeSS model, as it is involved through line management or the leadership group in both decisions regarding designing the team and its organizational context and setting the overall direction.

5.2 Team Alignment in Large-Scale Agile Projects

The reasons for limiting team authority in the studied cases were not because the organization used an authoritarian management style (the elements of which we have also found), but to ensure that all the teams involved in the large-scale development efforts achieved the expected outcomes towards the shared goals. (Kirsch et al. 2010) suggest that organizations may achieve alignment and ensure goal attainment through two major control modes: formal (behaviour and outcome) and informal control (clan and self-control) modes. Formal, top-down control is exercised by relying on documentation, written rules, and procedures that require particular behaviours to be followed in order to achieve the desired outcomes and could be related to managerial control. Informal or bottom-up control encourages groups and individuals to monitor their own work, which they are able to do if group and personal goals are largely compatible with the goals of the broader organization. Such control can be exercised as a collective action in what is called a clan, the members of which socialize and agree upon a set of norms and values, which empowers the clan members (Ouchi 1979).

We found that in Project Mobility teams were influenced by top-down alignment directives, while teams in Project Cloud exercised bottom-up alignment mechanisms, in which the community of teams can be understood as a clan. As in previous research that demonstrated that teams that are not involved in designing the general policies and alignment initiatives might not fully commit to those initiatives (Moe et al. 2009), we found that the top-down approach in Project Mobility was not suitable for all teams, and especially not for the experienced teams that constantly reflected upon and improved their own process. The experienced teams perceived the offered best practices (as proposed by the management) as a threat to their ability to make good decisions regarding their work. Our findings are in accordance with (Conboy and Carroll 2019) who found that a top-down approach to implementing processes in large-scale agile environments achieved mixed success because many members felt that the processes were imposed by those who did not understand the implications or problems to be solved. Alignment enforced by external stakeholders reduces team authority and is, therefore, a barrier to team autonomy in large-scale agile environments.

In contrast to the largely formal control mechanisms in Project Mobility, the bottom-up approach in Project Cloud emerged over time, as management allowed and encouraged teams to take more responsibility. A bottom-up approach to aligning joint efforts confers a higher potential for team empowerment and increased autonomy since team members participate in decision-making. The results of these initiatives can thus take not only the form of a "one-size-fits-all-solution," as is often the case in formal rules- and procedures-oriented control, but also consider initiatives tailored to the specific needs of the teams. Therefore, this alignment approach might better support teams in large-scale agile environments, in which the needs of each team can vary substantially. Empowerment in Project Cloud was established through mutual adjustments by team representatives in the community, a new informal governance unit comprising all teams working in the project, product owners, and technical experts. Notably, line management was not a part of the community. While individual teams in this project did not have full authority in all the levels defined by Hackman (1986), we believe that having a strong influence on the team design and the overall system direction as part of the community satisfies the needs of the teams and, at the same time, the needs for alignment held by the organization. It is also important to

note that the community was eligible to make the decisions, and not just recommend the decisions for approval by management. At the same time, we found that neither the teams nor the community in Project Cloud had much influence on team specialization and competence management. Therefore, although the level of authority given to the community as a unit is above the self-management level, it does not fulfil the criteria for a self-designing or a self-governing unit. However, the results of our analysis could help the project reflect on the areas where community authority is limited and decide whether more authority should be granted to raise the level of community autonomy to satisfy team requirements for further autonomy.

5.3 Identifying the Sweet Spot for Team Autonomy and Organizational Control

Dingsøyr et al. (2018) found that the ability to balance a decentralised decision process and a centralized structure was an important success factor for large-scale projects. Similarly, our data indicate that neither full autonomy of individual teams nor full autocracy of management seems to be a valid approach to authority distribution in large-scale agile development. Full autonomy of individual teams is unrealistic because teams in large-scale organizations do not deliver their work outcomes alone, and alignment is needed to ensure quality and avoid jeopardizing interdependencies (Olsson Holmström and Bosch 2016). Management-taken decisions in large and complex contexts can easily fail to recognize the needs and the complexity of the work performed by the individual teams, making at least some management-led alignment efforts viewed as unnecessary and discouraging by the teams.

So, what is the alternative? Our findings demonstrate how the decentralization of authority can be achieved by fostering the formation of communities for shared bottom-up decision-making, which at the same time fulfils the need for a centralized structure and alignment. This is in line with Kirsch et al. (2010) who found that in knowledge-intensive work, where tasks are less repetitive and more ambiguous, or in an environment where innovation and creativity are required, clan control is an appropriate and strong control mechanism for organizations. In our case, the community in Project Cloud can be regarded as a clan, where empowered members take a wide range of decisions together and then control the teams and individuals within that community. As all teams are involved in the clan, the teams largely control themselves, with the exception of decisions that affect the whole community. Similar clan or community control mechanisms have been found in three organizations that foster self-management (Lee and Edmondson 2017). Notably, the level of decentralization in the three cases studied by Lee and Edmondson differs, with the level of bottom-up authority growing proportionally with the size of the organizations (the largest organization had the lowest level of decentralization). Thus, the strategy of balancing decentralised authority with centralized structures seems to ensure the alignment necessary for handling the complexity and dependencies inherent in large-scale software development while leveraging the benefits of team autonomy. While it would be premature to claim that this represents an ideal "sweet spot" between the two opposing forces, we do think that this balance struck in Project Cloud, and supported by the LeSS and Spotify models, might approximate one. Indeed, it is possible that no single sweet spot exists, but that the right balance between formal control and team autonomy is context-dependent and could vary, e.g., based on organizational culture, the maturity of the agile implementation, the skill level of developers, attributes of the

product and its markets, etc. Exploring this further through additional empirical studies could be fruitful.

5.4 Implications for Practice and Theory

Our study generates a number of findings with practical implications. With respect to formal control, we found that:

- Enforcement of formal control can be problematic, especially for well-functioning and experienced teams, and lead to discouragement and a lack of commitment to such initiatives;
- The need for formal control changes over time, depending, for example, on the maturity of the teams.

For managers of large-scale agile projects searching for ways to increase the level of team authority without jeopardising the needs for alignment, we recommend finding ways to foster the mechanisms of clan control, which are suggested as an appropriate control mechanism for knowledge-intensive, less repetitive, and more ambiguous work, and when innovation and creativity are required (Kirsch et al. 2010):

- Facilitate the formation of strong organizational communities.
- Coach community members in creation of shared mental models, developing trust and connections.
- Foster communities with decision-making authority.

Besides the practical implications, our work has implications for theory. We used Hackman's model (Hackman 1986) to study authority distribution in the context of large-scale development efforts. The model explains most of our observations but does not sufficiently model the situation when authority is shared. As in Lee and Edmondson (2017), who have used Hackman's work to study the decentralization of decisions in organizations, our findings show that decision-making authority can be partially decentralized or shared, and we have further specified whether the decisions are made by the teams, shared by the teams with management or supporting roles, or made in a community setting. In addition, Hackman did not describe certain important factors, such as changes over time or the situational nature of authority distribution. Thus, this study shows that Hackman's model (Hackman 1986), although useful, needs to be amended to portray better the complexity of authority distribution in large-scale agile settings.

5.5 Limitations and Threats to Validity

In this paper, we sought to understand how the conflict between the teams' quest for autonomy as understood in the context of agile software development and the organizational quest for alignment and control associated with the large-scale manifests in industrial settings. This was done by conducting a multiple-case study of two large-scale software development projects at Ericsson, in which multiple teams developed the same product.

Our empirical enquiry is descriptive in nature and resulted in a detailed analysis of the authority distribution. In the following, we discuss the limitations and threats to the validity of our findings according to the definitions of validity and reliability proposed by Yin (2009).

Internal *validity* is concerned with the validity of causal relationships, and it is only relevant to explanatory or causal results (Yin 2009), and therefore not relevant in our case, as this research is exploratory in nature.

The construct *validity* of a case study concerns the identification of the correct operational measures for the concepts being studied (Yin 2009). The main threat to the construct validity of this research was the definition of what constitutes team autonomy and organizational control in large-scale software development projects. To do so, we have used Hackman's authority matrix as the theoretical ground (Hackman 1986), which differentiates between the area of responsibility of a work unit (autonomy) and the area of responsibility of the management (control), using four different areas of responsibility. One potential threat to the construct validity in our work is related to the accuracy of our interpretation of Hackman's authority matrix in the software engineering context, i.e., the types of decisions included in each area of responsibility, and the multi-team project context, i.e., the additional horizontal unit-to-unit relationship in addition to the vertical unit-to-management dimension included in the original authority matrix. We have suggested concrete types of decisions in Table 3, based on the definitions of the areas of responsibility and examples given by Hackman (1986) (see also Table 1). Our suggestions are in accordance with the work by Lee and Edmondson (2017), who studied decentralization of authority in three companies (two of which were software companies) and based their empirical work on Hackman (1986) and Puranam et al. (Puranam et al. 2014). The authors argue that decision authority can be centrally held by managers or decentralized in the following domains: (1) personnel and performance management; (2) managing and monitoring work execution; (3) work execution; (4) work and resource allocation; (5) organization and work design; and (6) firm strategy. Our interpretation of Hackman's model overlaps (with respect to responsibility areas) and extends (with respect to the types of decisions) the interpretation by Lee and Edmondson (2017), and the differences in the interpretation of the concepts are motivated by the different object of study (teams in our enquiry and organizations in theirs). To increase the validity of our observations, we particularly relied on data and investigator triangulation. We held joint workshops with the four authors to discuss the interpretation of the Hackman model cooperatively and to construct a set of definitions that were used to gather and analyse the interview data. The construct validity could also be affected by how the interviewees understood the studied phenomenon. To address this, we explicitly did not include questions regarding the levels of autonomy, but instead asked questions regarding different decision situations relevant to the daily duties of the interviewees. Notably, our findings are limited to the types of decisions included in the study, and repeating the study in new contexts could require adding new or changing the types of decisions relating to the four areas of responsibility, or additional new or changing the very areas of responsibility, for example, as done by Lee and Edmondson (2017). To ensure the construct validity of our own findings, we have reported the results at the level of individual decision types and have avoided aggregating the data based on authority levels or the areas of responsibility (Table 4).

The *external validity* of a case study concerns the domain to which the results can be generalized, which is known as the analytical generalization of a case study (Yin

2009). First, we would like to note that we studied two individual cases and thus generalize by similarity and variety, as opposed to statistical properties in sample-based generalizations (Wieringa and Daneva 2015). In the context of our study, external validity relates to the degree of support for the generalization of our findings with respect to the distribution of authority in large-scale agile software development projects. Our findings suggest that authority in large-scale settings in a number of responsibility areas can rest with the teams or with the management, as also found by Hackman (1986), or with a community, as also found by Lee and Edmondson (2017). Notably, we do not claim that the authority will be or shall be distributed in a particular way. The variability of the real world, the main problem of external validity (Wieringa and Daneva 2015) means that in other contexts, the actual way the authority is distributed may differ, as it differs in the two studied cases. However, we believe that the observed community-based mode of decision-making as an alternative to team-based or management-led decisions could have relevance and practicality (Wieringa and Daneva 2015) for similar projects. Based on our study, we can hypothesize that the theoretical population relevant in our context can be characterised by the following three aspects. First, in both studied cases, the systems under development were large, multifaceted, and technically demanding. Second, both case organizations had transformed their software development from a waterfall-type process to agile development. Third, the number of development teams was relatively large. In other words, the community-based authority could be of particular interest for organizations developing large-scale complex software systems involving many development teams, which have transitioned from traditional waterfall development to agile ways of working. It is worth noting that although both studied cases were distributed across multiple locations, we have not studied the authority distribution in all sites and cannot make specific claims as to whether the authority is distributed similarly or differently. Further research is needed to understand the nature of authority distribution across multiple locations. Finally, to increase the analytical generalizability of our descriptive findings, we have provided rich contextual descriptions of each of the studied cases to allow analytical comparisons and synthesis with other studies on similar subjects, as suggested by Yin (2009).

The *reliability* of research concerns the repeatability of the research (Yin 2009). If other researchers conducted the same study, would they arrive at the same findings? The main threat to the reliability (Yin 2009) of this research is the variability in the data collection. Data collection was conducted using the general interview guide approach (Patton 2014), which introduced variability to the topics discussed in the interviews. Our results could also be subjected to further reliability threats in terms of the reliability of the data collected through the interviews. To address this threat, we triangulated the data sources by recruiting a sufficiently "wide and varied" set of interviewees, i.e., a (large) set containing interviewees holding different roles in both cases (Patton 2014). We also triangulated the investigators by conducting all interviews with two interviewers cooperatively. The total number of interviews for Project Cloud was 82, with four focusing specifically on the division of authority in the project. The low number of interviewees with only one representative for each role can be considered a weakness, as we could not triangulate the results for each specific role. However, all responses from the interviewees in different roles were in agreement with respect to how authority was divided in the project, making us believe that adding more interviewees would have been of limited value. In Project Mobility, the number of interviews and interviewees was larger: eight interviews with 14 interviewees. In Project Mobility, the interviews were part of a larger longitudinal study in which a total of 28 interviews and seven workshops with teams were conducted, and

thus we acquired a good background knowledge of this case. Further, we have verified our case description with the representatives of the project, who have acknowledged that our interpretation with respect to authority distribution is accurate. Based on the aforementioned triangulation, we are confident that our findings, especially regarding Project Mobility, accurately reflect the perceptions of the members of the organizations.

6 Conclusions

The recent trend towards agile transformations at both the development and corporate levels has challenged the common authority distribution principles to allow for more bottom-up decisions to emerge. However, how to best achieve a balance between the need for alignment and the associated quality and process control, and the need for teams to be autonomous remains an open question. In our multiple-case study, we confirmed that large-scale software development efforts require alignment as well as autonomy, and that autonomy changes over time. Methods for implementing alignment are wide and varied. Top-down alignment initiatives include the centralization of decision-making power in the hands of management, introduction of supervisory roles and functions, and mandatory processes and checklists. Bottom-up alignment initiatives include conferring decision-making powers to the teams, consensus-based decision-making processes, and the delegation of certain functions from the experts to the teams. At the same time, the freedom of individual teams working in a large-scale software development environment cannot be limitless due to the complex dependencies of the work with other teams and stakeholders. Thus, community-like structures, similar to clans structures (Kirsch et al. 2010), have emerged as alternative control mechanisms. Therefore, while pure top-down control may reduce individual team authority and morale, clan control satisfies the need of the individual teams to influence decisions and participate in the decision-making process while enabling non-authoritative alignment and control across the teams. Indeed, a single sweet spot may not exist for team autonomy and organizational control. The right balance between formal control and team autonomy might be context-dependent and vary, e.g., based on organizational culture, the maturity of the agile implementation, developers' skill level, and attributes of the product and its markets. Further studies should be conducted to better understand various ways of balancing the need for alignment and autonomy and the impact on the context on how to do it successfully.

Appendix

Interview Guide for Project Mobility

Introductory questions:

- What is your role in the organization?
- Which decisions are you involved in?
- What do you think are the key current enablers (today) of efficiency in the organization?
- What do you think are the key current obstacles to efficiency in the organization?
- How would you describe the balance of control and autonomy?

Questions about the decision-making processes:
Structure

- Who do you coordinate your activities with?
- Who is involved in decisions you are responsible for?
- Are such decisions taken in different arenas (by different people)?
- Are similar decisions taken in remote sites as well? Is this done independently or in a coordinated fashion?
- How are decisions (taken in different arenas and/or in different sites) aligned? What happens if they are not aligned?
- How are conflicting priorities handled?

People

- Are decisions taken without informing people that should have been involved?
- Are all necessary roles involved in the decision-making? Anyone missing?
- Are there too many people involved in the decision-making process? Any redundancy?
- Are people from remote sites involved in decision-making?
- (if applicable) Can teams make these decisions, and if so, under which circumstances (team maturity, team composition)?

Information

- Do you have enough information to make the decisions?
- Do you receive any input when making decisions?
- Do you reach out to the remote sites when information is needed?

Practices:

- How can decision-making be improved (speed, information input, people)?

Interview Guide for Project Cloud

Introductory questions:

- Personal background

  What is your current role and tasks in the organization?
  What were your previous roles and tasks in the organization?

- Environment where works

  What is your current team/organization?
  How do your team collaboration with the rest of the organization?

Communities of practice:

- Which are the most important CoPs that participate?
- How do the CoPs work currently? (meetings, wiki, etc.)
- On which topics do the CoPs (that participates) make decisions?

– How are the decisions made?

Executing the team task:

– Who decides how to execute team tasks?
– Who designs features?
– Who develops features?
– Who performs quality control functions?
– Who delivers features?

Monitoring and managing ways of working and progress:

– Who monitors and manages ways of working and progress?
– Who defines team processes?
– Who can change and improve team processes?

Designing the team and its organizational context

– Who designs the team?
– Who decides who is on the team?
– Who is able to remove people from the team?
– Who links teams with supporting roles?

Setting overall direction:

– Who sets the overall direction?
– Who decides what shall be in the system?
– Who designs the product architecture?
– Who chooses specialization and work items for teams?

Change of authority over time:

– Has the level of team autonomy changed over time, and if so, why?
– Is the given level of authority sufficient?

Acknowledgements We are grateful to Ericsson and the interviewees from both projects for their sincere interest and continuous support.

Funding Open access funding provided by SINTEF AS. This research is partially funded by the GoLD project and the Swedish Knowledge Foundation under the KK-Hög grant 2016/0191, by the A-team project and the Research Council of Norway under the grants 235359, and by TEKES as part of the Need for Speed research programme of DIMECC (Finnish Strategic Center for Science, Technology and Innovation in the field of ICT and digital business).

References

Barney HT, Moe NB, Dybå T, Aurum A, Winata M (2009) Balancing individual and collaborative work in agile teams. In: International Conference on Agile Processes and Extreme Programming in Software Engineering. Springer, pp 53-62

Beck K (2000) Extreme programming explained: embrace change. addison-wesley professional

Berczuk S, Lv Y (2010) We're all in this together. IEEE Softw 27:12–15

Bick S, Spohrer K, Hoda R, Scheerer A, Heinzl A (2017) Coordination challenges in large-scale software development: a case study of planning misalignment in hybrid settings. IEEE Trans Softw Eng 44:932–950

Boehm B, Turner R (2005) Management challenges to implementing agile processes in traditional development organizations. IEEE Softw 22:30–39

Brown SL, Eisenhardt KM (1995) Product development: Past research, present findings, and future directions. Acad Manag Rev 20:343–378

Bäcklander G (2019) Doing complexity leadership theory: How agile coaches at Spotify practise enabling leadership. Creativity Innov Manag 28:42–60

Cohen SG, Bailey DE (1997) What makes teams work: Group effectiveness research from the shop floor to the executive suite. J Manag 23:239–290

Conboy K, Carroll N (2019) Implementing large-scale agile frameworks: Challenges and recommendations. IEEE Softw 36:44–50

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, Falessi D, Power K (2019) Agile development at scale: The next frontier. IEEE Softw 36:30–38

Dingsøyr T, Moe NB, Fægri TE, Seim EA (2018) Exploring software development at the very large-scale: A revelatory case study and research agenda for agile method adaptation. Empir Softw Eng 23:490–520. https://doi.org/10.1007/s10664-017-9524-2

Farrow A, Greene S (2008) Fast & predictable a lightweight release framework promotes agility through rhythm and flow. In: Agile 2008 Conference. IEEE, pp 224–228

Fenton-O'Creevy M (1998) Employee involvement and the middle manager: Evidence from a survey of organizations. J Organ Behav 19:67–84

Gittell J (2006) Relational coordination: Coordinating work through relationships of shared goals, shared knowledge and mutual respect. Relational Perspectives in Organizational Studies: A Research Companion 74–94. https://doi.org/10.4337/9781781950548.00011

Guzzo RA, Dickson MW (1996) Teams in organizations: Recent research on performance and effectiveness. Annu Rev Psychol 47:307–338

Hackman JR (1986) The psychology of self-management in organizations. In: Pallack MS, Perloff RO (eds) Psychology and work: Productivity, change, and employment. American Psycological Association, Washington

Heikkila VT, Paasivaara M, Lasssenius C, Damian D, Engblom C (2017) Managing the requirements flow from strategy to release in large-scale agile development: A case study at Ericsson. Empir Softw Eng 22:2892–2936. https://doi.org/10.1007/s10664-016-9491-z

Hewitt B, Walz D (2005) Using shared leadership to foster knowledge sharing in information systems development projects. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences. IEEE, pp 256a-256a

Hoda R, Noble J (2017) Becoming agile: a grounded theory of agile transitions in practice. Paper presented at the Proceedings of the 39th International Conference on Software Engineering, Buenos Aires, Argentina

Hoda R, Noble J, Marshall S (2012) Self-organizing roles on agile software development teams. IEEE Trans Software Eng 39:422–444

Hoegl M, Parboteeah P (2006) Autonomy and teamwork in innovative projects. Hum Resour Manage 45:67

Ingvaldsen JA, Rolfsen M (2012) Autonomous work groups and the challenge of inter-group coordination. Human Relations 65:861–881

Kirsch LJ, Ko D-G, Haney MH (2010) Investigating the antecedents of team-based clan control: Adding social capital as a predictor. Organ Sci 21:469–489

Kitzinger J (1995) Qualitative research: introducing focus groups Bmj 311:299–302

Klakegg OJ, Williams T, Shiferaw AT (2016) Taming the 'trolls': Major public projects in the making. Int J Proj Manag 34:282–296. https://doi.org/10.1016/j.ijproman.2015.03.008

Kniberg H (2014a) Spotify Engineering Culture (Part 1). https://www.youtube.com/watch?v=Yvfz4HGtoPc

Kniberg H (2014b) Spotify Engineering Culture (part 2). https://www.youtube.com/watch?v=vOt4BbWLWQw

Langfred CW (2000) The paradox of self-management: Individual and group autonomy in work groups. J Organ Behav 21:563–585

Langfred CW (2007) The downside of self-management: A longitudinal study of the effects tf conflict on trust autonomy, and task interdependence in self-managing teams. Acad Manag J 50:885–900
Larman C, Vodde B (2016) Large-scale scrum: More with LeSS Addison-Wesley Signature Series (Cohn). Addison-Wesley Professional, p. 368
Lee MY, Edmondson AC (2017) Self-managing organizations: Exploring the limits of less-hierarchical organizing. Res Organ Behav 37:35–58
Leffingwell D (2018) SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises. Addison-Wesley Professional
Lewis J, Neher K (2007) Over the waterfall in a barrel-MSIT adventures in scrum. In: Agile 2007. IEEE, pp 389–394. https://doi.org/10.1109/AGILE.2007.45
Mankins M, Garton E (2017) How Spotify balances employee autonomy and accountability. Harv Bus Rev 95.1
Mathieu J, Maynard MT, Rapp T, Gilson L (2008) Team effectiveness 1997–2007: A review of recent advancements and a glimpse into the future. J Manag 34:410–476. https://doi.org/10.1177/0149206308316061
Moe NB, Aurum A, Dybå T (2012) Challenges of shared decision-making: A multiple case study of agile software development. Information and Software Technology 54(8):853–865
Moe NB, Dahl B, Stray V, Karlsen LS, Schjødt-Osmo S (2019) Team autonomy in large-scale agile. In: Proceedings of the 52nd Hawaii International Conference on System Sciences
Moe NB, Dingsøyr T, Dybå T (2009) Overcoming barriers to self-management in software teams. IEEE Softw 26:20–26
Moe NB, Dingsøyr T, Dybå T (2010) A teamwork model for understanding an agile team: A case study of a Scrum project. Inform Software Tech 52:480–491. https://doi.org/10.1016/j.infsof.2009.11.004
Morgan G (2006) Images of organization. SAGE Publications, Thousand Oaks
Olsson Holmström H, Bosch J (2016) No more bosses?: A multi-case study on the emerging use of non-hierarchical principles in large-scale software development. In: Product-focused software process improvement: 17th international conference PROFES 2016. Springer, pp 86-101
Ouchi WG (1979) A conceptual framework for the design of organizational control mechanisms. Manage Sci 25:833–848
Patton MQ (2014) Qualitative research and evaluation methods. Sage
Pearce CL (2004) The future of leadership: Combining vertical and shared leadership to transform knowledge work. Acad Manag Perspect 18:47–57
Petersen K, Wohlin C (2010) The effect of moving from a plan-driven to an incremental software development approach with agile practices. Empir Softw Eng 15:654–693
Puranam P, Alexy O, Reitzig M (2014) What's "new" about new forms of organizing? Acad Manag Rev 39:162–180
Paasivaara M, Lassenius C (2014) Communities of practice in a large distributed agile software development organization – Case Ericsson. Inform Softw Technol 56:1556–1577
Paasivaara M, Lassenius C (2019) Empower your agile organization: Community-based decision making in large-scale agile development at ericsson. IEEE Softw 36:64–69
Paasivaara M, Lassenius C, Heikkila VT, Dikert K, Engblom C, Ieee (2013) Integrating Global Sites into the Lean and Agile Transformation at Ericsson. 2013 Ieee 8th International Conference on Global Software Engineering. Ieee, New York. https://doi.org/10.1109/icgse.2013.25
Paasivaara M, Lassenius C, Heikkilä VT (2012) Inter-team coordination in large-scale globally distributed scrum: Do scrum-of-scrums really work? In: Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement. pp 235–238
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, Ireland
Saldana J (2009) An introduction to codes and coding. The coding manual for qualitative researchers. Sage, Thousand Oaks
Beedle SK (2001) Agile Software Development with Scrum. Prentice Hall, Upper Saddle River
Semler R (1989) Managing without managers. Harv Bus Rev 67:76–84
Smite D, Moe N, Floryan M, Lavinta G, Chatzipetrou P (2020) Spotify Guilds: When the Value Increases Engagement, Engagement Increases the Value. Communications of the ACM
Smite D, Moe NB, Levinta G, Floryan M (2019) Spotify guilds: how to succeed with knowledge sharing in large-scale agile organizations. IEEE Softw 36:51–57
Šmite D, Moe NB, Šāblis A, Wohlin C (2017) Software teams and their knowledge networks in large-scale software development. Inf Softw Technol 86:71–86
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
Schwaber K, Sutherland J (2017) The scrum guide. Scrum Alliance
Schwaber K, Sutherland J (2020) The scrum guide. Scrum Alliance

Trist EL, Bamforth KW (1951) Some social and psychological consequences of the longwall method of coal-getting: An examination of the psychological situation and defences of a work group in relation to the social structure and technological content of the work system. Hum Relat 4:3–38
Wenger E, McDermott RA, Snyder W (2002) Cultivating communities of practice: A guide to managing knowledge. Harvard Business Press
Wieringa R, Daneva M (2015) Six strategies for generalizing software engineering theories. Sci Comput Program 101:136–152
Yin RK (2009) Case study research: Design and methods. Sage, Thousand Oaks

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

Darja Šmite is a full professor of software engineering at the Blekinge Institute of Technology, where she leads research efforts and education on global software development. She has led a number of nationally funded research projects related to the effects of offshoring for the Swedish software industry. Her other research interests include large-scale agile software development and software process improvement. Šmite received a Ph.D. in computer science from the University of Latvia. She holds a research scientist position at SINTEF and a part-time adjunct professorship at NTNU.

Maria Paasivaara is a professor of empirical software engineering at LUT University and an Adjunct Professor at Aalto University. Her research interests include software engineering processes and practices, agile software development, large-scale agile, software project management, global software engineering and software engineering educational research. She performs empirical research in close collaboration with industrial and academic partners and aims at solving real-world problems that are important to the software industry. She has a D.Sc. degree from Helsinki University of Technology.

Casper Lassenius is an associate professor at Aalto University, Helsinki, Finland, and an adjunct research scientist at the Simula Metropolitan Center for Digital Engineering, Oslo, Norway. His research interests include software engineering management, with a focus on agile methods and organizational transformations. Lassenius received an M.Sc. in industrial management, and a Dr.Sc. (Tech.) in software engineering, both from Helsinki University of Technology.