Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Henrik Forbord

Balancing Alignment and Autonomy in Large-Scale Agile Software Development

A Mixed-Methods Exploratory Case Study

Master's thesis in Computer Science
Supervisor: Torgeir Dingsøyr
July 2022

NTNU
Norwegian University of
Science and Technology

Henrik Forbord

Balancing Alignment and Autonomy in Large-Scale Agile Software Development

A Mixed-Methods Exploratory Case Study

Master’s thesis in Computer Science
Supervisor: Torgeir Dingsøyr
July 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

The success of agile development methods in small teams has inspired the use in large software development settings. However, applying agile at scale has proven challenging. In large-scale software development, many teams need to work together to coordinate their efforts. Prior studies report on challenges related to communication overload, external distraction, absence of transparency and alignment across teams, and lack of team autonomy. In particular, finding the optimum balance between alignment and teams' willingness to be autonomous remains an unresolved issue. I conducted a mixed-methods case study in a large-scale software development firm to provide contextually relevant advice for practitioners on how to achieve this balance. The firm had adopted a tribe structure and implemented several communities of practice for alignment and decision-making. Based on an analysis of 13 interviews, 35 hours of observations, a questionnaire with 31 respondents, and a variety of supplemental material, I present several findings on inter-team coordination and team autonomy.

I found that the teams were partly autonomous. Every team defined and customized their internal work processes, whereas team autonomy on software requirements and architectural decisions was neither expedient nor desirable from the developers' point of view. For alignment, I found that communities of practice with decision-making authority functioned effectively. The communities let teams align on tools and best practices, and they fostered relationships and increased transparency levels. Further, I found that team synchronization demands dedicated formal arenas. Specifically, I suggest project-specific team leader meetings. The latter was lacking in the case organization.

From my findings, I offer the following three contributions. First, I identify three practices for mitigating team synchronization and transparency challenges at scale: guide teams within the same product or feature area into sub-groups; decentralize decision-making by fostering communities with decision-making authority in specific areas; facilitate small inter-team meetings for teams having joint goals. Second, I present a team autonomy model on four dimensions relevant to software development teams in multi-team environments. The four dimensions are processes, requirements, architecture, and tasks. Processes relate to the team-internal agile way of working. Requirements refer to functional and non-functional software requirements. Architecture pertains to local architectural and technical choices. Tasks capture the how and who of task execution. Third, and finally, I demonstrate how the team autonomy model can be used to understand what decisions teams influence.

Keywords— Team synchronization, Transparency, Autonomy, Self-management, Inter-team coordination, Large-scale agile, Software development, Software engineering, Case study

Sammendrag

Bruken av smidige metoder i små team har vist seg å være suksessfult, hvilket har inspirert til bruk også i større programvareutviklingprosjekt. Dette har imidlertid vist seg å være utfordrende. I storskala programvareutvikling må mange team samarbeide for å koordinere arbeid. Tidligere studier rapporterer om utfordringer knyttet til kommunikasjonsoverbelastning, ekstern distraksjon, fravær av åpenhet og synkronisering på tvers av team, og mangel på teamautonomi. Særlig har det vist seg vanskelig å balansere koordinasjon av team med hvert enkelt teams ønske om å være autonom. Jeg gjennomførte et kombinert kvalitativt og kvantitativt casestudie hos et storskala programvareutviklingsfirma for å gi praktikere kontekstrelevante råd for hvordan man best oppnår denne balansen. Firmaet hadde implementert en tribestruktur, samt flere praksisfellesskap for koordinasjon og beslutningstaking. Basert på en analyse av 13 intervjuer, 35 timer med observasjoner, et spørreskjema med 31 respondenter og en del tilleggsmateriale, presenterer jeg med denne masteroppgaven flere funn på inter-team-koordinasjon og teamautonomi.

Jeg fant at teamene var delvis autonome. Alle teamene definerte og tilpasset sine interne arbeidsprosesser, hvorpå teamautonomi vedrørende programvarekrav og arkitekturavgjørelser så ut til å være hverken formålstjenelig eller ønskelig fra utviklernes ståsted. Hva gjelder koordinasjon, fant jeg at praksisfellesskap med beslutningsmyndighet fungerte effektivt. Praksisfellesskapene lot teamene koordinere bruken av verktøy og gode utviklingspraksiser, og de fostret kollegiale forhold og økt åpenhet. Videre fant jeg at teamsynkronisering krever dedikerte formelle arenaer. Mer konkret foreslår jeg prosjekt-spesifikke teamledermøter. Dette manglet hos caseorganisasjonen.

Oppgaven har tre bidrag. Først presenterer jeg tre praksiser som reduserer utfordringer knyttet til teamsynkronisering og åpenhet i storskala programvareutvikling: lokaliser team som opererer innenfor samme produktområde eller leverer tjenester innenfor samme funksjonalitetsområde i sub-grupperinger; desentraliser avgjørelser gjennom praksisfellesskap med beslutningsmyndighet på spesifikke områder; og fasiliter små møtearenaer for team med felles mål. Bidrag nummer to er en teamautonomi-modell med fire dimensjoner relevant for programvareutviklingsteam i miljø med mange team. De fire dimensjonene er prosess, programvarekrav, arkitektur og arbeidsoppgaver. Prosess viser til teaminterne smidige prosesser. Programvarekrav referer til funksjonelle og ikke-funskjonelle krav. Med arkitektur menes lokale programvarearkitekturvalg og andre lokale tekniske valg. Arbeidsoppgaver innbefatter hvordan oppgaver løses og hvem som løser dem. Til sist, som et tredje bidrag, demonstrerer jeg hvordan teamautnomi-modellen kan benyttes til å forstå hvilke avgjørelser programvareteam påvirker.

Preface

Sadly, but not so sadly, this master's thesis, written between January and July 2022 by the undersigned (aka me), concludes my master's degree in Computer Science at the Norwegian University of Science and Technology (NTNU). What started with an introductory course to information technology, examen philosophicum, discrete mathematics, and single variable calculus fall 2017 has come to an end.

This thesis follows a preparatory specialization project on large-scale agile software development written last fall. The specialization project included a literature review, and it aroused my curiosity and interest in software development efforts at scale. I therefore decided to dedicate my last semester at NTNU to studying this topic further.

Some people enabled me to write this thesis. First and foremost, I would like to thank my supervisor Professor Torgeir Dingsøyr at NTNU for his valuable guidance throughout the spring and last fall. His weekly feedback and suggestions have guided me steadily toward the goal of this master's thesis. I also wish to thank Signicat and the informants for their willingness to share their experiences. I especially thank those who took the time to participate in interviews. Further, I would like to thank my fellow master hall students for the (too) many hours of procrastination. Last but not least, I would like to thank my girlfriend and all who made my time at NTNU in Trondheim and Gløshaugen so memorable. Thank you.

Henrik Forbord
Trondheim, 15th July 2022

Contents

List of Figures

List of Tables

Acronyms

CoP	community of practice. 10, 12, 13, 17, 30, 31, 41, 44–49
CPM	country product manager. 24, 27, 31, 32, 43–45
DA	disciplined agile. 1
DevOps	development and operations. 27, 31, 36
eID	electronic identification. 23, 24, 26, 27, 32, 37, 43–45
KPI	key performance indicator. 26, 27, 30–32, 37, 45
LeSS	Large-scale Scrum. 1, 45
PM	product manager. 24, 25, 31–33, 47
PO	product owner. 6, 10, 24, 25, 27, 31, 32, 36–38, 43, 45–47
QA	quality assurance. 25, 31, 36, 38
S@S	Scrum-at-Scale. 1, 45
SAFe	Scaled Agile Framework. 1, 12, 13
SM	scrum master. 6, 24, 27, 32, 33, 37, 46, 47
SoS	Scrum of Scrums. 9, 10, 12, 45, 46, 49
UX	user experience. 25, 31, 34, 37, 38
WIP	work in progress. 7
XP	eXtreme programming. 6

1 Introduction

In this chapter, I introduce agile and large-scale agile before I present indicated challenges with crucial aspects of agile software development: alignment and autonomy. I also illustrate why scaling agile seems to be a thorny issue for practitioners. Further, I present the goal and research question of the thesis, and I also manifest the scope and target audience. Lastly, I outline the overall structure of the thesis. —

1.1 Problem Statement

Since the formulation of the agile manifesto in 2001 (Beck et al.), agile methods (e.g., Scrum and Kanban) have transformed the way software is developed by emphasizing short iterations of work, adaptability and flexibility, active end-user involvement, and tolerance to change (Dingsøyr et al., 2012). Traditional approaches assuming information systems are fully specifiable and best built sequentially with extensive up-front planning have been nearly entirely eradicated (Sridhar Nerur et al., 2005). In the latest State of Agile (2021), 94 % of the respondents report their company is practicing agile. Additionally, the report sees an explosive increase in agile adoption across functions of enterprises. The success of agile methods in small software development projects has turned attention towards using such methods at scale (Dingsøyr et al., 2019), despite being criticized for being applicable only to small teams (Dybå and Dingsøyr, 2008).

Several approaches to scaling agile have been proposed over time. The first generation of large-scale agile methods combined advice from agile practices and project management frameworks such as the Project Management Body of Knowledge. Later on, large-scale agile frameworks such as Scaled Agile Framework (SAFe), Scrum-at-Scale (S@S), Large-scale Scrum (LeSS), disciplined agile (DA) and the Spotify model have evolved to support agile software development at scale. This master's thesis reports a case study of an organization that has adopted and is adjusting certain elements similar to core elements of the Spotify Model. The model, introduced by Kniberg and Ivarsson (2012), has become influential among agile proponents and has formed the basis of agile methods used in several organizations (Salameh and Bass, 2018, 2021), though only 5% of practitioners, according to the latest State of Agile, works in an organization following the Spotify model. In the report, SAFe significantly outdistances the Spotify model and other frameworks as the prominent scaling framework, being used by more than a third of the respondents. However, SAFe is a comprehensive system dedicated to scaling agile, whereas the Spotify model is not trademarked as a model. In the 12th State of Agile (2018), the Spotify model was not even included as a scaled method but as an agile method and practice along with Scrum, Kanban, and other agile method variants. There may therefore be reasonable to consider the number of practitioners using the Spotify Model to scale agile to be under-reported.

The term "large-scale development" has several definitions and interpretations. Barlow et al. (2011) define the term as characterized by multiple stakeholders and complex projects within large organizations, whereas Rolland et al. (2016) suggests that large-scale involves complex integration with various internal and external information systems and multiple stakeholders with different interests. This master's thesis however applies the definition by Dingsøyr et al. (2018a) which defines it as "development efforts that involve a large number of actors, a large number of systems and interdependencies, which have more than two teams."

Scaling agile raises many challenges, including a lack of alignment among teams and issues regarding decision-making efficiency (Dingsøyr et al., 2019). Applying agile at scale threatens fundamental assumptions of agile. Self-management is one of the core principles of the agile manifesto, and providing teams with the autonomy to self-organize is considered a success factor at scale (Dikert et al., 2016; Edison et al., 2021). Self-managing teams offer some advantages over traditionally managed teams because bringing decision-making authority to the operational level increases the speed and accuracy of problem-solving (Dingsøyr et al., 2018a; Moe et al., 2009). However, self-management reduces the ability to coordinate across teams (Ingvaldsen and Rolfsen, 2012). While teams need to self-manage, teams working towards the same goal require much coordination and management effort (Petersen and Wohlin, 2010). The more complex software projects get, the more difficult they become to control and manage. According to Malone and Crowston (1993), coordination is "the managing of dependencies". Intricate and numerous dependencies complicate the coordination of multiple teams, and this calls for processes for managing and aligning inter-team decisions. In this thesis, I use the inter-team coordination taxonomy by Berntzen et al. to identify inter-team coordination mechanisms at Signicat. The taxonomy includes three categories of mechanisms that contribute to sharing information and knowledge about the software development process, namely meetings, roles, and tools and artifacts.

An increasingly distributed workforce may amplify inter-team coordination challenges, and the outbreak of COVID-19 has accelerated this trend. According to the latest State of Agile report, only 3 % (!) of the 4182 respondents indicated that they plan to return to office full time, and since the 7th State of Agile (2013), the number of practitioners that works in a company with distributed agile teams has increased from 35% to 89% (see Figure 1.1). This calls for coordination mechanisms that do not rely only on co-location but rather embrace other characteristics. I use the TOPS framework (Berntzen et al., 2022) to illustrate the underlying characteristics of the identified inter-team coordination mechanisms at Signicat.

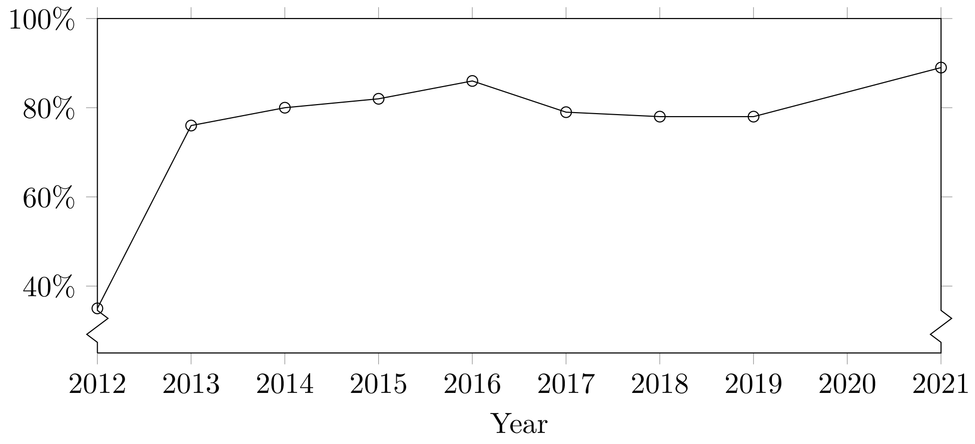


Figure 1.1: Software practitioners working in organizations with distributed teams for the last ten years according to the State of Agile. No report in 2020 due to COVID-19

1.2 Goal and Research Question

In 2012, Paasivaara et al. (2012) asked for more empirical research on how to tackle inter-team coordination in large-scale agile projects. Still, in 2018, achieving inter-team coordination was highlighted by Bick et al. (2018) as one of the most pressing challenges in large-scale software development. Dingsøyr et al. (2018a) called for further investigation on how to balance autonomy with the need for alignment and organizational control after exploring software development at the very large-scale, and more recently, Moe et al. (2021) asked for further studies to investigate various ways of balancing the need for alignment and autonomy. As new frameworks increase in popularity, Dingsøyr et al. (2019) ask researchers to provide contextually relevant advice for practitioners on how to introduce scaling frameworks. With this case study, I seek to contribute to this. Most recently, Moe et al. (2021) helped deepen the understanding of what kind of authority teams should have by analyzing the autonomy of agile teams in an organization using Hackman's classification of unit authority. However, *how* to best achieve a balance between the need for alignment and each team's need to be autonomous remained an open question of Moe et al., which is a gap I aim to contribute to filling by addressing the following research question:

How can alignment and autonomy be balanced in large-scale agile software development?

1.3 Contributions

With this master's thesis, I move the research one step closer to identifying how to balance alignment and autonomy in large-scale agile by providing the following three contributions:

1. *Three identified practices for mitigating team synchronization and transparency challenges at scale: guide teams within the same product or feature area into sub-groups (tribes); decentralize decision-making by fostering communities (guilds) with decision-making authority; facilitate small inter-team meetings for teams having joint goals.*
2. *A team autonomy model on four dimensions relevant to software development teams: process, requirements, architecture, and tasks.*
3. *An actionable approach to use the team autonomy model by introducing four statements that map to each of the dimensions in the model.*

I further elaborate on the contributions in the concluding chapter of this master's thesis (Chapter 6).

1.4 Scope and Target Audience

Due to a limited time scope of six months (January-July), I have scoped this master's thesis to only discuss the main findings against a few of the challenging areas large-scale agile faces. I will also emphasize that I have written this thesis with a software engineering lens, simplifying some organizational aspects of the case organization. I exclude departments such as sales and marketing when outlining the organizational structure, and I omit the role of managers and C-executives. As the daily business and regular deliveries happen via the software development teams, I argue that the most relevant details regarding the organizational structure are brought to the table from the lens of software engineering by providing an overview of the work context in which the teams operate within on a daily basis.

I have set researchers with a particular interest in large-scale agile software development as the targeted audience of this master's thesis. Specifically, researchers interested in process improvement and agile transformations will likely find the issues discussed in this thesis interesting. I also target software project managers and agile coaches practicing agile at scale. Additionally, the thesis may be an example study of large-scale agile for computer science students curious about the topic.

Primarily, the design of the master's requires the reader to have some experience with software development to fully understand the potential intricate and complex processes software development (at scale) involves. It is an advantage to have prior knowledge of agile software development, but not necessarily as Chapter 2 should present the fundamental principles behind agile. However, suppose you are a novice computer science student or just interested in the topic without certain experience in computer science and agile methodologies. In that case, I strongly recommend reading the chapters chronologically with special attention to the theory chapter (Chapter 2). As an experienced researcher or software practitioner within large-scale agile, you can dip into any chapter of interest. However, I recommend you to start by getting familiar with the TOPS framework presented in subsection 2.2.1 as this frames most of the analysis. Further, I recommend you to read the research critique section (Section 3.4) and the case description section (Section 4.1) in order to obtain an understanding of the limitations of the research design and contextual factors of the study before diving into the the results (Chapter 4) and discussions (Chapter 5).

1.5 Thesis Structure

The remainder of this article is structured as follows. Chapter 2 presents relevant background theory to the reported findings in Chapter 4 and the elaborations and discussions in Chapter 5.

Chapter 3, the method chapter, outlines the research design of the master's thesis, the chosen strategies, and data collection methods. It also describes the analyzing approaches I have utilized. The method chapter also highlights significant factors limiting or strengthening the study.

In Chapter 4, I present a new autonomy model and report the identified findings. I also present the case context in depth.

In Chapter 5, I discuss my main findings, present implications for research and practice, and discuss and evaluate the potential limitations of my findings.

Finally, Chapter 6 concludes the thesis. Here I bring a conclusion to the research question of the thesis, and I also outline my contributions and suggestions for future work.

2 Background Theory and Related Work

Software development at scale has proven to be challenging because it demands several teams to coordinate their efforts (Moe et al., 2016). A central question is whether or not one should apply and adapt agile methods to such environments (Dingsøyr et al., 2019). In this chapter, I first describe the differences between traditional and agile software development before presenting two agile methods in detail, Scrum and Kanban. Second, I provide different views on what constitutes large-scale before presenting the first approaches to scaling agile. I also provide an overview of some of the most common large-scale agile software development frameworks used in the industry, particularly the Spotify model. Further, I present coordination and team-related challenges in large-scale agile (Edison et al., 2021), before I focus on two critical aspects in agile software development at scale: inter-team coordination and team autonomy. Finally, I introduce a new team autonomy model for large-scale agile software development.

2.1 Agile Software Development

Traditional approaches to software development, for instance, the Waterfall method, emphasize intense upfront planning, documentation, and sequential implementation and deployment (Sridhar Nerur et al., 2005), which builds upon an assumption that software is fully specifiable prior to start developing it. Contrary to this, agile approaches assume that software can be built iteratively in small increments based on feedback and continuous learning and adaptation (Dingsøyr et al., 2012).

With agile, the literature report benefits on customer collaboration, work processes for handling defects, and estimation. On the other hand, agile approaches are repeatedly criticized for lacking support for design and architecture (Dybå and Dingsøyr, 2008). But what exactly does it mean to be agile, and what are the key characteristics of agility when developing software? In 2001, a group of software developers formulated the agile manifesto, highlighting interactions, working software, customer collaboration, and responding to change as important factors (Beck et al., 2001). However, there has not been a clear and unified theoretical understanding of agile software development until recently. Most definitions provided in the literature since 2001 highlight, in different ways, the ability to create and respond to change as the underlying core of being agile (Highsmith, 2002; Conboy, 2009; Agile Alliance, 2021). In 2002, Highsmith defined agile as *"the ability to both create and respond to change in order to profit in a turbulent business environment"*. In 2009, Conboy argued that agility means to *"rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value*, and the Agile Alliance has defined agile as *"the ability to create and respond to change"*. More recently, Baham and Hirschheim (2021) suggested four core concepts and facets of agile software development:

1. **Incremental design and iterative development:** Anticipating change by working iteratively in short delivery cycles, thereby reducing the product scope to small increments to create opportunities for inspection; Creating change through incremental software design in response to change from what has been learned.

2. **Inspect and adapt cycles:** Anticipating change by instituting ceremonies for inspecting and adapting (i.e., learning from and creating change in response to discovered changes) the product increment (e.g., simplifying design, testing software frequently) and the development process (e.g., updating work statuses, reevaluating team processes, reprioritizing requirements).

3. **Working cooperatively/Collaboratively/In close communication:** Anticipating change through recognizing and predicting changes in one's environment; Creating change as a team by working together to respond to change from what has been learned collectively.
4. **Continuous customer involvement:**In addition to the cell above, centralizing user requirements changes by working together with the customer to collectively identify and respond to change early through close customer involvement.

Since the formulation of the Agile Manifesto 2001, several agile methods have been proposed. According to the 15th State of Annual (2021), Scrum[1], Kanban[2], or a mix of them (Scrumban), are the most popular ones, with Scrum as the overwhelmingly most popular with 66% of the respondents identifying it as the methodology they follow most closely. Examples of other methods addressing core principles of the Agile Manifesto are eXtreme programming (XP)[3], Lean startup[4] and Crystal methodologies[5]. Albeit, they seem to be significantly less popular and on the brink of being eradicated: XP and combinations of Scrum and XP have dropped from 30 % use in the third State of Agile to less than 7 % today, and only 1% are now using pure XP; 1 % only uses Lean startup, and the most recent report merged Crystal methodologies into a pot of "others" methods in with a total of 5 %. In contrast, the most significant growth trend has been the usage of Kanban boards for workflow management and visualization. This approach is now used by 77% of respondents, heavily increasing from 6% in the very first State of Agile, and is now the most used practice for agile planning regardless of method implementation.

The trend of applying Scrum or Kanban, or a mix of them, is also reflected in the case organization studied in this master's thesis. Scrum and Kanban are the only two methods applied at the team level, and almost all teams use Kanban boards. Consequently, I focus on those two methods in the following.

2.1.1 Scrum

Schwaber and Sutherland developed the Scrum framework in the early 1990s aiming at small software development projects. However, its usage has extended to environments of multiple teams (Dingsøyr et al., 2019), becoming what is known as Scrum at Scale (later presented in Section 2.2). Scrum established the scrum master (SM) role and product owner (PO) role. Along with developers, those roles compose a *cross-functional* and *self-organizing* Scrum team - meaning the Scrum team has all the skills necessary to create value and manage their work. Within such a team, no sub-team exists, and hierarchies are absent. The SM is a process facilitator, and the PO is accountable for maximizing product and business value. Central in Scrum is rather three *artifacts* representing work or value: a product backlog, a sprint backlog, and an increment. Whereas the product backlog is the ordered list of all work remaining to be undertaken by a Scrum team, a sprint backlog is a subset of the product backlog selected for a *sprint* - a time-boxed iteration of fixed length (one month or less, typically). During a sprint, a Scrum team develops a product *increment*, a concrete stepping stone toward a final product. Another artifact some Scrum teams use is the burndown chart - a graphical representation of the completed work in a sprint and the total work remaining in the same sprint. The sprint is the heart of Scrum, figuring as a container of the four other *events* of Scrum: sprint planning; daily Scrum; sprint review; and sprint retrospective. A new Sprint starts immediately after the conclusion of the previous Sprint. Typically, a Scrum team makes no changes during a sprint. Instead, they can cancel a sprint if the sprint goal becomes obsolete. Only the PO has the authority to cancel the Sprint. I present an overview of a Sprint in Figure 2.1, which also present the four other events in more detail. The SM drives all the events during a sprint. The events facilitate Scrum teams to discuss requirements, identify potential dependencies, estimate efforts, and reflect on recent experiences to fine-tune their processes. However, research on Scrum has shown that this may be challenging (Moe et al., 2010), particularly in distributed environments (Hossain et al., 2009). Though the Scrum framework provides a set of clearly defined roles, events, and artifacts, (Hron and Obwegeser) suggest tailoring Scrum to its specific needs and contextual circumstances. This may include employing

[1]https://www.scrum.org/resources/what-is-scrum
[2]https://kanban.university/
[3]http://www.extremeprogramming.org/
[4]http://theleanstartup.com/
[5]https://alistair.cockburn.us/

processes, techniques, and methods that are not part of Scrum per se, for instance, by applying elements from Kanban or implementing elements from traditional non-agile project management frameworks.

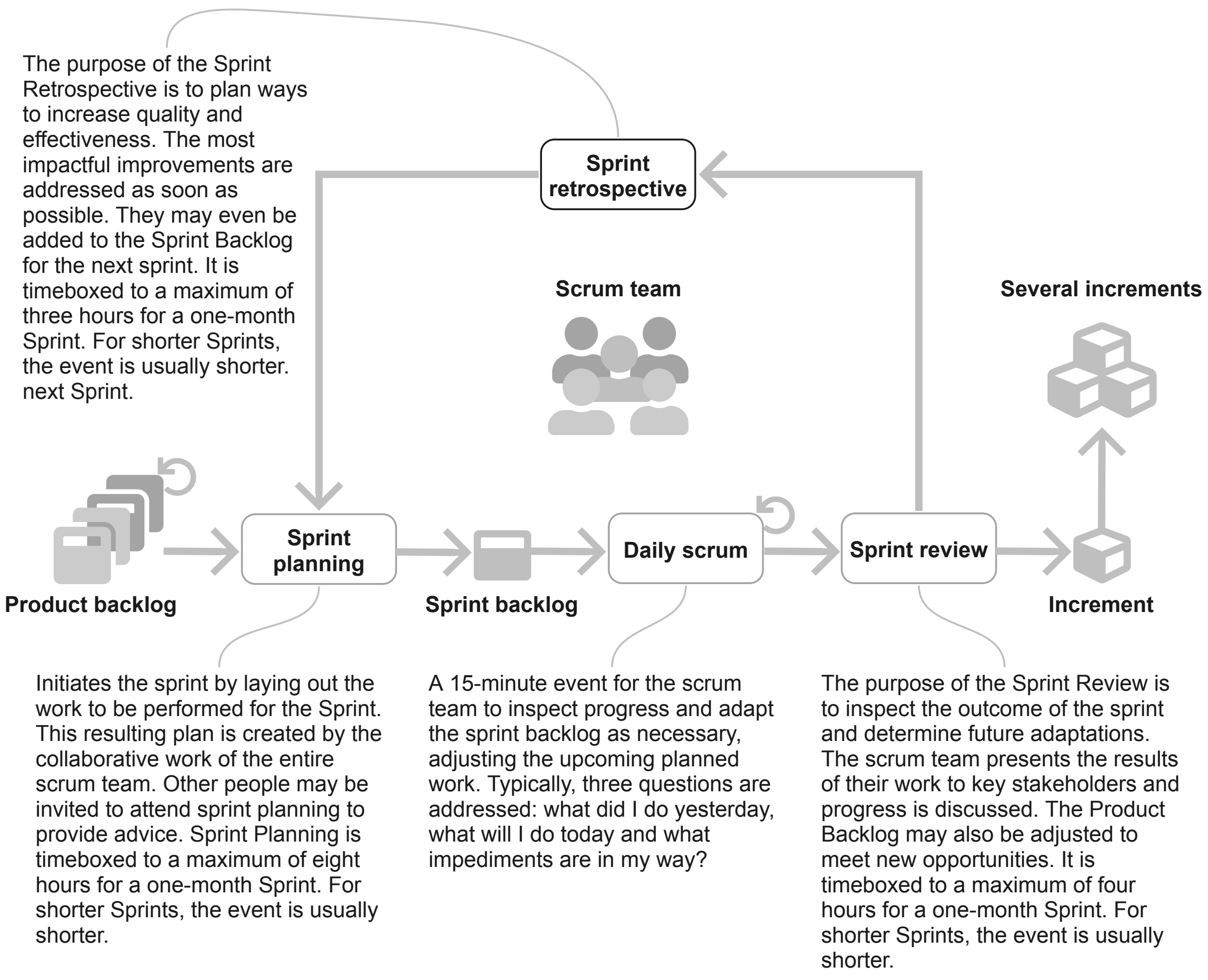

The purpose of the Sprint Retrospective is to plan ways to increase quality and effectiveness. The most impactful improvements are addressed as soon as possible. They may even be added to the Sprint Backlog for the next sprint. It is timeboxed to a maximum of three hours for a one-month Sprint. For shorter Sprints, the event is usually shorter. next Sprint.
Sprint retrospective
Scrum team
Several increments
Product backlog
Sprint planning
Sprint backlog
Daily scrum
Sprint review
Increment
Initiates the sprint by laying out the work to be performed for the Sprint. This resulting plan is created by the collaborative work of the entire scrum team. Other people may be invited to attend sprint planning to provide advice. Sprint Planning is timeboxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter.
A 15-minute event for the scrum team to inspect progress and adapt the sprint backlog as necessary, adjusting the upcoming planned work. Typically, three questions are addressed: what did I do yesterday, what will I do today and what impediments are in my way?
The purpose of the Sprint Review is to inspect the outcome of the sprint and determine future adaptations. The scrum team presents the results of their work to key stakeholders and progress is discussed. The Product Backlog may also be adjusted to meet new opportunities. It is timeboxed to a maximum of four hours for a one-month Sprint. For shorter Sprints, the event is usually shorter.

Figure 2.1: The Scrum framework

2.1.2 Kanban

What differentiates Kanban from Scrum and other methods is that Kanban is neither a methodology nor a process framework. According to the Kanban University, it is a management method or approach that should be applied to an existing process or way of working. Kanban does not require the creation of any new roles or events (Dingsøyr et al., 2019) but is instead all about visualizing work, limiting work in progress (WIP), and maximizing efficiency, which the *Kanban board* facilitates. Within a Kanban board, one pulls work from left to right: new work items enter the board on the left. When they exit on the right, customers should have received value. Work items of a Kanban board can be of different types and sizes, and board designs may vary greatly, often including columns more than only "to do," "doing," and "done." In order to maximize flow, the WIP limit is key, serving as a constraint of work that is allowed to enter the different columns of the board. However, Kanban does not specify what the WIP limit should be (Sjøberg et al.). Setting correct WIP limits is perceived as challenging and has taken more time than expected for teams implementing the Kanban method (Senapathi and Drury-Grogan, 2020). One of the ground pillars of Kanban is that completed work is more valuable than starting new work: too much ongoing work may create bottlenecks and reduce developer efficiency, negatively affecting the work *flow*.

Compared to Scrum, the central idea of Kanban is continuous and fluid processes rather than short and structured sprints with a start and end. A comparison of Scrum and Kanban is provided in Table 2.1.

	Scrum	**Kanban**
Idealogy	Cross-functional, self-organizing teams delivering small increments and learning through experiences	Visualize work to ensure WIP is limited such that a continuous flow can be maintained
Cadence	Fixed-length sprints (typically 2-4 weeks)	Continuous flow
Practices	Sprint planning, daily Scrum, sprint review and sprint retrospective	Visualize work, limit WIP, manage flow, implement feedback loops, make policies explicit, and improve collaboratively, evolve experimentally
Roles	Scrum master and product owner	No required roles
Artifacts	Product backlog, sprint backlog and increment	Kanban board

Notes: WIP=Work In Progress.

Table 2.1: Comparison of Scrum and Kanban

2.2 Large-scale Agile Software Development

Despite being criticized for being applicable only to small teams (Dybå and Dingsøyr, 2008), the popularity of agile has expanded well beyond small-team projects and into the large-scale. The topic "large-scale agile" appears to originate in the mid-2000s (Hoda et al., 2018). Eckstein's book Agile Software Development in the Large, published in 2004 was the first on the topic. Despite the growing interest in applying agile methods to large projects (Freudenberg and Sharp, 2010), there is no single definition in the literature of what constitutes large-scale agile. Researchers have suggested several interpretations over the years: a minimum number of developers (e.g., 50 developers); a minimum number of teams (e.g., five teams); a minimum lines of code (e.g., ½ million lines); or a minimum of different time zones covered by the development organization (e.g., three zones). In 2014, Dingsøyr et al. proposed a taxonomy of scale of agile software development projects, defining large-scale as agile development efforts with two to nine teams, and very large-scale as agile development efforts with more than nine teams. Figure 2.2 illustrates the taxonomy. As mentioned in the introductory chapter (Chapter 1), this master's thesis applies this definition.

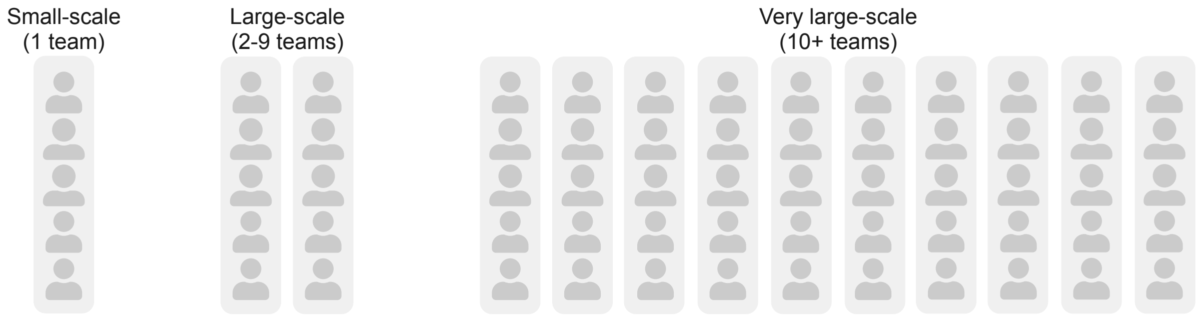


Figure 2.2: A taxonomy of scale for agile software development (Dingsøyr et al., 2014)

However, scaling agile is a significant change for many organizations (Dingsøyr et al., 2019). It causes the rise of several challenges as it brings along difficulties such as an increased number of stakeholders, a higher number of teams that need to coordinate their efforts, an added system and software complexity, and an increasingly intricate architecture (Badampudi et al., 2013). In "Comparing Methods for Large-scale Agile Software Development: A Systematic Literature Review," Edison et al. (2021) identified 31 challenges associated with the five large-scale agile methods and grouped them into categories. Understanding how to avoid all those challenging categories and their sub-challenges is key to harnessing the benefits of agility on a large scale. However, according to the goal and scope of this master's thesis, which is to provide answers on how to balance alignment with team autonomy, this thesis focuses on *inter-team coordination* and *team-related* challenges.

2.2.1 Inter-team Coordination

In large-scale agile software development, well-functioning collaboration within each team is not sufficient to develop desired software solutions. As one scales up, the complexity of the developed software increases, and several teams need to work together (Rolland et al., 2016). Previous studies have addressed challenges associated with the lack of synchronization and alignment among teams, (Bick et al., 2018), and at Ericsson, maintaining transparency across the high number of development teams and projects has seen to be a demanding challenge (Paasivaara et al., 2018).

The increased complexity at scale and the high number of involved stakeholders often result in multiple and possibly intricate dependencies not necessarily present in smaller development efforts. According to Strode's dependency taxonomy (Strode, 2016), dependencies occur when the completion of a task or an action relies either on the output of a previous task or action (process dependency), the presence of some artifact or person (resource dependency), or when a project requires a form of information to progress (knowledge dependency): process dependencies consist of activity and business process dependencies; resource dependencies include entity and technical dependencies; and knowledge dependencies comprise requirement, expertise, historical, and task-allocation dependencies. When dependencies are well managed, appropriate coordination mechanisms are in place to support the smooth flow of interdependent work. Poorly managed dependencies indicate that the chosen coordination mechanisms are inappropriate, inadequate, or absent. Furthermore, unmanaged dependencies can constrain or block progress leading to delay as people wait for resources, for the activities of others to be completed, or for necessary information.

In order to manage such dependencies between teams, *inter-team coordination* is necessary (Strode, 2016). In the Performe progamme at the Norwegian Public Service Pension Fund (the "Pension Fund"), Dingsøyr et al. (2018a) found that many arenas for coordination of work and knowledge sharing made it easy for team members to adjust to activities of other teams. This included arenas such as Scrum of Scrums (SoS) and open work area. SoS are a form of the daily stand-up meeting practice in Scrum where "ambassadors" represent their teams (Agile Alliance, 2022), and open work area typically allows teams to sit together in an open-plan office space on one floor.

Extant literature presents several taxonomies, frameworks, and mechanisms to categorize coordination within software development. Strode et al. (2011) distinguish between explicit and implicit coordination. The first consist of mechanistic and organic elements such as plans, rules, routines, mutual adjustment, and feedback, whereas the latter consist of cognitive elements like shared mental models and team expertise. Based on this, Scheerer et al. (2014) conceptualize mechanistic elements as centralized top-down planning with vertical coordination, e.g., between a team and a hierarchically superior person or a central team, and organic elements as decentralized bottom-up adjustments with horizontal coordination between teams. In organizational science, earlier contributions to coordination include Ven et al. 's (1976) three coordination modes, differentiating between coordination by feedback (personal and group mode) and coordination by plans, procedures, and tools (impersonal mode). A more novel approach to categorize coordination is the taxonomy and TOPS framework proposed by Berntzen et al. (2022), in which is the taxonomy and framework I use to analyze (Chapter 4) the coordination mechanisms present at Signicat (for the reasoning of this choice, see Section 5.5 in the discussion chapter). Therefore, I choose to present this taxonomy and the TOPS framework in more detail.

TOPS framework

The taxonomy of inter-team coordination mechanisms proposed by Berntzen et al. (2022) contributes to the dependency taxonomy of Strode (2016) by extending the focus to the inter-team level. The taxonomy includes **three categories** that includes a total of *six sub-categories*:

- **Meetings:** Scheduled and un-scheduled, such as SoS and ad-hoc chats.
- **Roles:** Individual and teams, such as program architects and platform teams.
- **Tools and artifacts:** Tangible and intangible, such as Slack and physical task boards.

In total, the taxonomy identify 27 inter-team coordination mechanisms (see Berntzen et al., 2022, pg. 11).

Coordination meetings contribute to sharing information and knowledge about product and development processes and provide inter-team representatives with access to information held by expert roles (e.g., an architect), hence managing knowledge, process, and resource dependencies. Examples of meetings are CoP meetings and PO workshops (e.g., for planning and discussing longer-term technical product-related areas).

Coordination roles may manage resource, knowledge, and process dependencies. Their overview of technical and business process dependencies makes them important assets for developers to resolve dependencies across teams. Contrary to previous significant studies of inter-team coordination in large-scale agile (Dingsøyr et al., 2018a; Moe et al., 2018; Dingsøyr et al., 2018b; Paasivaara and Lassenius, 2019; Stray and Moe, 2020), Berntzen et al. also include certain team roles as coordination mechanisms. For instance, they show how platform teams help assure technical alignment by providing product teams with common platform level services, thus managing resource dependencies. Other examples of coordination roles are program architects and dedicated test teams.

Coordination tools and artifacts are the third and last category of the taxonomy and may be tangible, material, or intangible, digital entities. Examples of tools are communication tools such as Slack and documentation tools such as Confluence, both being intangible tools for managing dependencies related to the development process. Slack supports communication and enables frequent and timely knowledge sharing across locations, whereas Confluence may store several intangible artifacts such as burndown charts and task boards (e.g., Kanban boards). A task board may also be a tangible entity (e.g., represented on a whiteboard in an open space area).

As introducing a scaling framework is a significant change for many (Dingsøyr et al., 2019), a central question is how to implement and adapt different inter-team coordination mechanisms. The technical-organizational-physical-social (TOPS) framework proposed by the same Berntzen et al. lets organizations assess different mechanisms' underlying characteristics. The four characteristics of TOPS are defined as follows:

- **Technical characteristics** refers to coordination mechanisms related to managing dependencies related to the software product. It also applies to digital tools or platforms supporting the development process. For example, the architect role or the tool Slack.

- **Organizational characteristics** refers to coordination mechanisms that capture the wider structural context of the development organization, managing business process dependencies in particular. For instance, team design and organizational design.

- **Physical characteristics** refers to spatial or tangible characteristics of a coordination mechanism. For instance, intangible mechanisms with spatial dependencies and physical artifacts and objects such as task boards

- **Social characteristics** refers to interpersonal or community-based characteristics of a coordination mechanism related to the management of interpersonal dependencies. For example, roles or activities that enable coordination through groups, typically a meeting.

These characteristics help organizations choose the mechanisms that address their coordination needs. For instance, implementing a large number of mechanisms requiring physical presence may not be adequate for distributed environments, and the early phases of a development programme may benefit from having a series of mechanisms with social characteristics to establish communication (Dingsøyr et al., 2018a).

2.2.2 Team Autonomy

Self-organization (i.e., autonomy) in software development is recognized as one of the central principles behind the Agile manifesto, which states that *"the best architectures, requirements, and designs emerge from self-organizing teams"*. Team-level autonomy refers to teams being provided freedom in carrying out tasks within their organization (Langfred, 2007). However, maintaining team autonomy and aligning self-organizing teams when scaling agile has proven to be challenging (Moe et al., 2016; Stray et al., 2018). Previous research has identified several barriers to autonomous teams in large-scale agile. For example, Scrum lacks guidelines for team organizing (Moe et al., 2008), and the Spotify model does not provide guidelines or influential factors for aligning autonomous teams. Besides, there is a tension between teams' autonomy and alignment (Salameh and Bass, 2018, 2019), and the introduction of extra roles and agile arenas at the inter-team level negatively affects team autonomy (Dingsøyr et al., 2018a).

The right balance between formal control and alignment and team autonomy are context-dependent and may vary based on organizational culture, developers' skill level, and attributes of a product and its markets (Moe et al., 2021). Also, the need for formal control may change over time, depending, for example, on the maturity of a team or an organization. Several studies recommend fostering communities with decision-making authority to increase the level of team autonomy without jeopardizing the need for alignment (Paasivaara and Lassenius, 2014; Salameh and Bass, 2018, 2021; Paasivaara and Lassenius, 2019; Moe et al., 2021), but research on the amount of team autonomy that is appropriate at scale remains an unresolved issue.

A few studies (Moe et al., 2021; Lee and Edmondson, 2017) have investigated authority distribution in large-scale contexts using the work of Hackman (1986). According to Hackman, there are four types of organizational authorities that need to be handled, and four types of teams depending on which of the authorities they have decision authority: a manager-led team lets the team decide on how to execute tasks only, whereas a self-managing team has authority to execute its tasks, as well as to monitor and manage its internal work process and progress; self-designing teams also get to design the team and its context, while a self-governing team has the authority to set the overall direction as well.

However, as outlined by Moe et al., the work of Hackman origins from the airline industry and not the software development industry, and though his studies were conducted in large-scale settings, the framework is not straightforward to use in the context of large-scale agile software development. With this master's thesis, I seek to begin the work of developing a model suited for measuring team autonomy in large-scale agile software development. I propose a model designed as a Kiviat chart (see Figure 2.3), which quantitatively measures levels of team autonomy in four areas (named *dimensions* in the model), namely on architecture, process, requirements, and tasks. The four dimensions are defined as follows:

- **Requirements:** Identifying, shaping, and prioritizing software requirements (what shall be in the system)
- **Architecture:** Deciding on local architectural decisions (all kinds of local technical choices)
- **Process:** Defining, changing, and improving internal agile framework, ceremonies, and processes (how to work)
- **Tasks:** Deciding how tasks should be executed and solved (how and by who)

This is in line with how Moe et al. understood the four functions of Hackman 's (1986) authority matrix in software development companies: when investigating the allocation of authority in large-scale agile, Moe et al. sought to address whom design product architecture and features; who defines, changes, and improves internal team processes; who decides what shall be in the system; and who develops features. As Moe et al., I use the expression "level of autonomy" to refer to the amount of authority held by a team, and the model measures the levels of team autonomy on a scale ranging from 1 to 5 (with 5 as the highest level of autonomy). How to derive the level of autonomy for a team with this model is presented in more detail in the method chapter (see Section 3.3). In the result chapter, I present the model with data from Signicat, the case organization, to demonstrate how the model can be used to understand what decisions development teams influence.

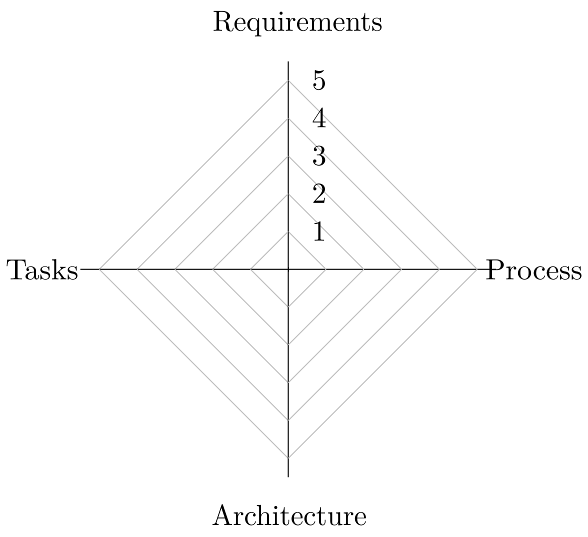


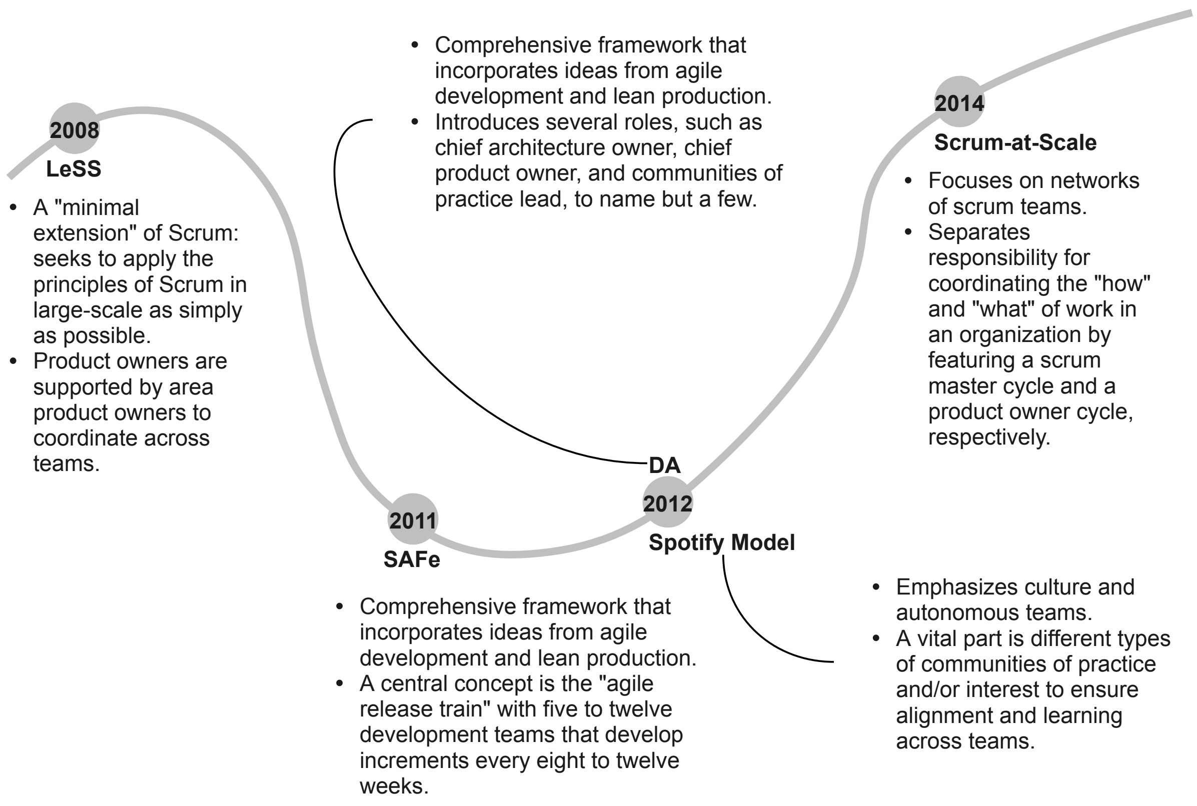
Figure 2.3: Team autonomy model on four dimensions

2.2.3 Scaling Agile

Several approaches to scale agile have been proposed over time. One of the earliest described practices to scale agile is the Scrum of Scrums (SoS) - a scaled form of the daily stand-up meeting practice in Scrum where "team ambassadors" represent their team (Agile Alliance, 2022). The format of the SoS is similar to Scrum's regular daily stand-up meeting. Each team representative presents work completed, the work to be conducted, and eventual impediments that need resolving. However, the practice has been reported as ineffective due to an often too wide audience to keep every participant interested and participants not knowing what to report that might be valuable to other teams (Paasivaara et al., 2012). Albeit, more successful variants such as weekly SoS meetings covering specific features, so-called feature SoS meetings (Paasivaara and Lassenius, 2014), and SoS meetings at sub-project level (Dingsøyr et al., 2018a) have been reported.

Another practice early proposed as a possible approach to scale agile is with Communities of Practice. A community of practice (CoP) is a group of people of a common interest or knowledge that meet regularly (e.g., once a week or monthly). Examples of such CoP include security CoP for software security and a Scrum Master CoP for sharing best practices on the agile way of working. In some cases, CoP have replaced the SoS approach, and shown to be a central mechanism behind successful large-scale agile implementations (Paasivaara and Lassenius, 2014). The formation of CoP can help achieve coordination and alignment, common design principles, organizational transparency, and knowledge sharing on best practices on several topics (Paasivaara and Lassenius, 2014). However, similar to SoS, implementing well-functioning communities is not easy, for instance, due to the lack of common activities and goals, persons at different levels, or participants at different time zones (Smite et al., 2019).

In response to the difficulties faced when scaling agile, several agile methods dedicated to large-scale have been proposed over the last decade, aiming at addressing large-scale development challenges. In Figure 2.4, I present an overview of the main five methods (Edison et al., 2021). The methods offer, to a different extent, specific sets of principles, practices, roles, tools, and metrics in order to scale agile. Whereas some of the methods are comprehensive frameworks providing extensive recommendations, such as the Scaled Agile Framework (SAFe), other methods, for instance, the Spotify Model, provide less rigidity and ceremony and instead emphasize a culture of distributed decision authority among self-organizing teams (Dingsøyr et al., 2019). Since Signicat, the case organization of this master's thesis has recently adopted and is currently tailoring a structure that evokes associations to some of the core elements of the Spotify model, I present that specific model and related literature in more detail in the following.



2008
LeSS
• A "minimal extension" of Scrum: seeks to apply the principles of Scrum in large-scale as simply as possible.
• Product owners are supported by area product owners to coordinate across teams.
• Comprehensive framework that incorporates ideas from agile development and lean production.
• Introduces several roles, such as chief architecture owner, chief product owner, and communities of practice lead, to name but a few.
DA
2011
SAFe
• Comprehensive framework that incorporates ideas from agile development and lean production.
• A central concept is the "agile release train" with five to twelve development teams that develop increments every eight to twelve weeks.
2012
Spotify Model
• Emphasizes culture and autonomous teams.
• A vital part is different types of communities of practice and/or interest to ensure alignment and learning across teams.
2014
Scrum-at-Scale
• Focuses on networks of scrum teams.
• Separates responsibility for coordinating the "how" and "what" of work in an organization by featuring a scrum master cycle and a product owner cycle, respectively.

Figure 2.4: The main five large-scale agile methods (Edison et al., 2021)

Spotify Model

The Spotify model was introduced to the world in 2012 by Henrik Kniberg and Anders Ivarsson when they published the whitepaper Scaling Agile @ Spotify (Kniberg and Ivarsson, 2012), which introduced the radically simple way Spotify[6], the largest and most popular audio streaming subscription service in the world, approached agile software development. The model is a people-driven approach for scaling agile that emphasizes the importance of culture and network rather than a specific set of practices. In traditional scaling frameworks, such as SAFe, specific practices tell how to execute the framework. In contrast, the Spotify model focuses on how businesses can structure organizations to enable agility. The Spotify Model is usually composed of four core elements, namely squads (commonly referred to as teams in the literature), chapters, and guilds (commonly referred to as CoP in the literature), and so-called tribes. This forms a variant of a matrix organization; several teams compose a tribe, representing the vertical structure of the matrix; whereas personnel of different teams composes chapters (within a tribe) and guilds (across tribes), representing the horizontal structure of the matrix (see Figure 2.5).

Each team is a cross-functional, autonomous team responsible for its way of working, and an agile coach (or Scrum Master) should support the team, and a product owner should guide it (Kniberg and Ivarsson, 2012). Tribes foster alignment across feature area-related teams, and a tribe is led by a tribe lead responsible for resource utilization and coordination across agile teams within the tribe and for helping teams collaborate with other tribes. Within each tribe, one can form chapters to allow personnel of different teams to align on best practices and solve problems within their expertise (e.g., a specific programming language or a specific frontend framework). The last core piece of the matrix is the guild. Whereas chapters belong to a tribe, guilds are wide-reaching groups of personnel across the whole organization with a desire to share knowledge and practice over the whole organization (Smite et al., 2019). As indicated, both chapters and guilds are model-specific variants of the CoP (previously described

[6]https://newsroom.spotify.com/company-info/

in Section 2.2), and they may vary in size, mission, membership, and activities.

However, a common tendency among the Spotify Model and the previously presented large-scale methods is to take greenfield settings for granted, which rarely is the case. Large, established software vendors often do not face legacy-free development and structure, and hence, most organizations confront challenges when applying large-scale agile development methods (Edison et al., 2021).

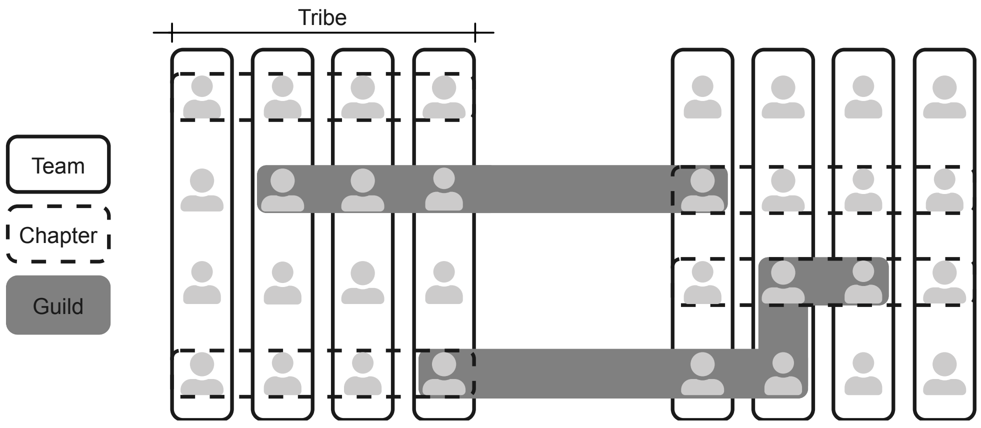


Figure 2.5: The organizational components of the Spotify Model's matrix structure

3 Research Method

To answer my research question, I used a mixed-method exploratory case study (Oates, 2006; Yin, 2014). I chose to perform an explorative case study because my goal is to provide new insights on how to balance alignment and autonomy in large-scale agile software development (see Section 1.2). In the following, I present the research process in detail. This includes insights on how I designed the case study and an introduction to the company I chose to study. I also present details on the data collection and data analysis. Finally, I critically assess my research by highlighting the limitations and strengths of the study design (Runeson and Höst, 2009).

3.1 Research Design

The company I chose to study, Signicat, is a leading provider of digital identity solutions in Europe. Signicat, established in Norway in 2007, now employs around 400 people around Europe. I chose this case because it involved many agile teams maintaining and developing a complex product portfolio, thus being well suited to explore inter-team coordination at scale. I present a thorough case description in the results (see Section 4.1).

My supervisor facilitated the initial contact with Signicat, and my supervisor and I met with Signicat representatives in November 2021. During this meeting, we learned more about the organization, the team structure, and the ongoing development projects. aIt became clear that the case represented a unique opportunity to study inter-team coordination in a fast-growing, large-scale agile software company with a complex and compound product portfolio with an extensive stakeholder group across Europe.

Following the November meeting, I articulated a paper elaborating on my thoughts for a possible case study at Signicat, including a sketch for data collection, which I sent to Signicat for consideration (see appendix A). I wanted to observe two teams in-depth, and to remove effects due to a particular way of working, I argued for observing teams operating in different product areas with different agile methods and processes. When the opportunity arose to conduct a case study in spring 2022, we scheduled another meeting in the middle of December 2021, and I commenced the data collection ultimo January 2022. I was designated one Scrum team and one Kanban team, operating in two different product areas at Signicat, just as requested. In the following section and subsections, I present details on the data collection.

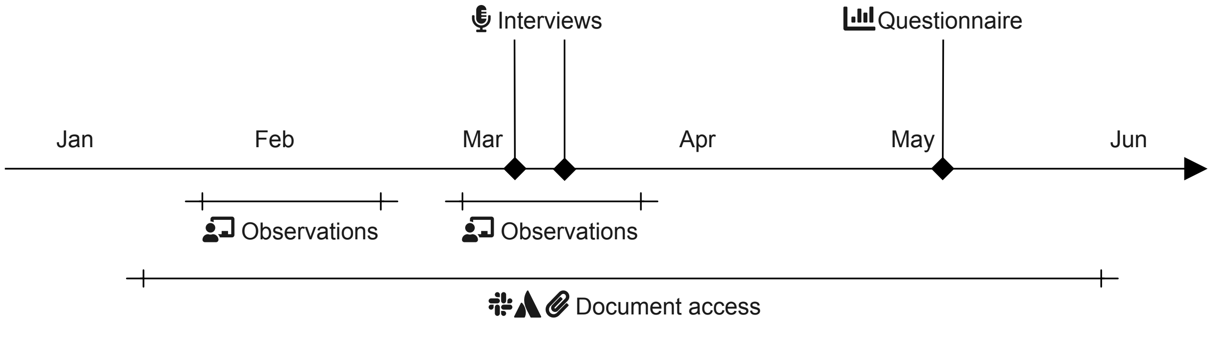


Figure 3.1: Timeline showing the data collection points. Document access includes access to Slack, Atlassian, and other supplemental material

3.2 Data Collection

The data collection occurred from January to May, and Figure 3.1 provides an overview of the data collection events during this period. To obtain rich and as detailed insights as possible, I utilize both qualitative and quantitative data collection methods and analysis. My collected data consists of meeting and ceremony observations of two teams, individual interviews, a questionnaire, and documents (a documentation database and a digital communication tool). Table 3.1 provides a more detailed overview of the collected data, and the following subsections present the different collection methods in more detail.

Data collection method	**Details**
Observations	Two periods of three weeks with observations of two teams' agile ceremonies and meetings. In total, 35 hours of observations, including: • 8 Stand-up meetings • 8 Planning meetings • 6 Prioritization meetings* • 2 Common retrospective meetings • 3 Demo meetings** • 1 Retrospective meetings
Interviews	13 interviews with a mean length of 27 minutes (11 on-site and 2 remote). Interviewees: • 5 developers (mean company tenure 3 years, mean IT tenure 8 years with a low of 2 years and a high of 31 years) • 1 scrum master • 2 product owners • 1 product manager • 2 tech leads • 1 tribe architect • 1 tribe lead Excluding the 5 developers: mean company tenure 4.5 years, mean IT tenure 22 years with a minimum of 4 years and a maximum of 40 years.
Documents	Access to documents January-June, including Slack logs and internal documentation on Confluence and Jira.
Questionnaire	Nine statements, with a five-level Likert scale, on team autonomy and inter-team coordination distributed to all tech sites across Europe. I collected 31 anonymous responses.

Notes: *backlog refinements, **including a company-wide variant.

Table 3.1: Data collection details

3.2.1 Observations

I observed the two agile teams I had been designated, including all their daily and weekly meetings and ceremonies for six weeks. I conducted the observations as non-participant observations with detailed notetaking, and I consider this appropriate to the research question I seek to answer. I wrote my observation notes following a predefined observation protocol (see appendix B). The protocol specified the record's content, participants present, description and purpose of the activity, direct quotes snippets of conversations, and my reflections on the observations. The extensive amount of detailed field notes resulted in a large data material of the observations. Figure 3.2 includes a snippet from a sample note from a team's stand-up meeting.

The observations allowed me to understand different people's behavior and actions in natural settings. I conducted the observations in two rounds, each round being three weeks, separated by a pause of three weeks. I split observations into two rounds to allow me to analyze data and see if some adjustments were necessary before conducting another round of observations. I made a few adjustments, such as asking to attend some other types of agile arenas (resulting in an invitation to a refinement meeting 22nd of March with several stakeholders I did not get to meet in the first round of the observations). Most of the observations were intra-team meetings within the two development teams. However, the intra-team meetings almost always covered inter-team aspects, which made them relevant to my analyses.

3.2.2 Interviews

I conducted 13 semi-structured individual interviews at Signicat's offices in Trondheim. I targeted informants from different areas of software development (i.e. personnel holding different roles) to obtain various perspectives (see Table 3.1). The interviews were conducted March 2022 (see Figure 3.1). Although the interviews were largely conversation-driven, I used an interview guide to guide the conversations. The full interview guide is provided in appendix C. Some standard questions asked were:

- What responsibilities do you have?
- Who decides on architecture?
- How do you manage dependencies?
- How do you share information across teams?

The interview guide was revised after the first six interviews to adapt the questions to provide more specific information on concerns emerging from the first six interviews. For instance, in the last seven interviews, I included more specific questions regarding the community of practice at Signicat.

I invited the interviewees via e-mail, and they were provided an information letter and consent according to the standards of the Norwegian center for research data a few days before the interview's scheduled date. I considered an interview length of 20-30 minutes appropriate to retrieve as much information as possible without being too time exhaustive such that someone avoided accepting an invitation. I transcribed the interviews verbatim for detailed analysis (further described in Section 3.3). Figure 3.2 includes an excerpt from an interview with a developer.

3.2.3 Documents

A third source of data is documents. Documents include material accessible within project management and communication tools used at Signicat (Confluence, Jira, and Slack). I had access to those tools throughout the data collection period (see Figure 3.1). Examining these sources provided me additional data for the analysis, for instance, information about team organization and documentation of the "way of working" of personnel and teams I did not observe nor interview. Figure 3.2 includes an extracted Slack chat log of a product owner and developer conversation.

3.2.4 Questionnaire

A questionnaire was used as a final data source to provide quantitative data. The questionnaire contained eight statements, four regarding team autonomy and four regarding inter-team coordination. The respondents indicated their degree of agreement on a five-level Likert scale. A ninth question let the respondents pick three improvement areas out of the eight areas covered in the eight statements. Two of the topics and statements were:

- Requirements: It is the team itself that identifies, shapes, and prioritizes the software requirements (what shall be in the system).
- Dependency awareness: We are aware of dependencies to other teams.

See Appendix D for the entire questionnaire.

To distribute the questionnaire, I e-mailed each tribe lead. Before distribution, Signicat representatives and my supervisor and I refined the questionnaire during a meeting. I also user-tested the questionnaire on a fellow master student to weed out ambiguities, which resulted in a reformulation of two statements and a redesign of the improvement area section of the questionnaire. I targeted all tech sites at Signicat, not just the two teams I got to know during my observations. Out of 110 agile team members, 31 responded to the questionnaire.

3.3 Data Analysis

Due to the chosen data collection methods' divergent nature, the analysis was quantitative and qualitative. I applied the quantitative analysis to the questionnaire results. First, the collected data were aggregated into histograms using PGFPlots[1] to illustrate the distribution of answers on all the eight statements provided in the questionnaire.

Further, to develop the autonomy model I briefly presented in the theory chapter (see Section 2.2.2), the five-level Likert scale of the questionnaire was translated into numerable equivalents to form a system suited for mean calculations: strongly disagree (=1); disagree (=2); neither agree nor disagree (=3); agree (=4); and strongly agree (=5). This allowed me to calculate mean values based on answers (*la*) from n respondents (see equation 3.1) on the team autonomy statements in the questionnaire such that I could score the level of team autonomy at Signicat. Each mean was rounded to one decimal place with a minimum value of 1 and a maximum value of 5. Further, as a final step in the quantitative analysis, I used the tkz-kiviat package[2] to design the autonomy model as a Kiviat diagram (also known as a radar chart or spider chart). I chose a kiviat design as this type of diagram is well suited for displaying multivariate data. To draw the model, I used PGFPlots functionality. The results chapter presents the model with mean values from Signicat respondents (see Section 4.2).

$$autonomy_{dimension} = \frac{1}{n}\sum_{i=1}^{n} a_n = \frac{a_1 + a_2 + \cdots + a_n}{n} \tag{3.1}$$

To analyze the interview data, I used qualitative direct content analysis as described by Schreier (2013). I systematically assigned segments of interview transcripts (phrases, sentences, or even whole paragraphs) to coding frames using different colorized markers. The inter-team coordination taxonomy developed by Berntzen et al. (2022) was chosen as the coding frame because I considered this taxonomy to be best suited for identifying inter-team coordination mechanisms in place at Signicat (see Section 5.5 for more details on this choice). I also used the same Berntzen et al.'s TOPS framework (see Section 2.2.1) to identify the underlying characteristic(s) of each potential inter-team coordination mechanism. As a third coding framework, I used the dependency taxonomy of Strode (2016) to define what dependencies each potential inter-team coordination mechanism managed.

However, as my research question is open-ended and explorative, and some of my data collection methods (the documents, as well as the observations to some extent) turned out to be better suited for a less systematic analysis, I decided to also investigate my collected data less rigid and without coding frameworks (Mayring, 2000). I continued the qualitative analysis by thematically grouping data from different data sources by linking them to three challenges at scale identified by Edison et al. (2021), namely team synchronization, transparency, and team autonomy. I outline these three challenges in Section 2, and they are further elaborated in Section 5 as they underpin the discussion of the main findings of this master's thesis. Figure 3.2 provides an illustrative example of the less rigid analysis approach.

After coding the interview material and thematically grouping items of retrieved data material, I wrote down what I considered the most interesting findings and made a slide presentation. I presented this to Signicat representatives of the two teams for checking and feedback. The meeting led to minor adjustments to the outlined findings and guidance for further data collection. Reactions to the findings included a broad agreement on the lack of dedicated coordination arenas among inter-related teams identified during the observation and interviews. I also held a final presentation in mid-June 2022 for Signicat representatives presenting the main findings from my analysis.

[1]https://ctan.org/pkg/pgfplots
[2]https://ctan.org/pkg/tkz-kiviat

Phases	Description
1: Familiarizing with the data	I transcribed the interview data after the thirteenth and final interview. I visited several Slack channels over numerous points in time during the spring. I took extensive meeting notes during all observations.
2: Coding the data	Along with reading transcriptions and meeting notes, I coded the material based on the taxonomy of Berntzen et al. (2021) to identify mechanisms used for inter-team coordination. I also used the TOPS framework (Berntzen et al., 2021) to identify the characteristics of each mechanism and Strode et al.'s (2016) dependency taxonomy to define which type of dependencies each mechanism managed.
3: Grouping the data	I linked data material from transcripts, meeting notes, and documents to the challenging areas of synchronization, transparency, and team autonomy (Edison et al., 2021). Figure 3.2 shows an example of how I did this. The figure also illustrates how the quantitative data from the questionnaire supplemented the qualitative data.
4: Reexamining the data	I re-examined the identified inter-team coordination mechanisms from step 2 to investigate whether or not they addressed some of the three challenges mentioned in step 3.
5: Reporting the data	I reported the findings. The outcome of step 2 formed the analysis (results section), whereas the outcome of step 3 and 4 lay the foundation for elaborating and discussing the findings (discussion chapter).

Table 3.2: Phases of the qualitative analysis.

3.4 Research Critique

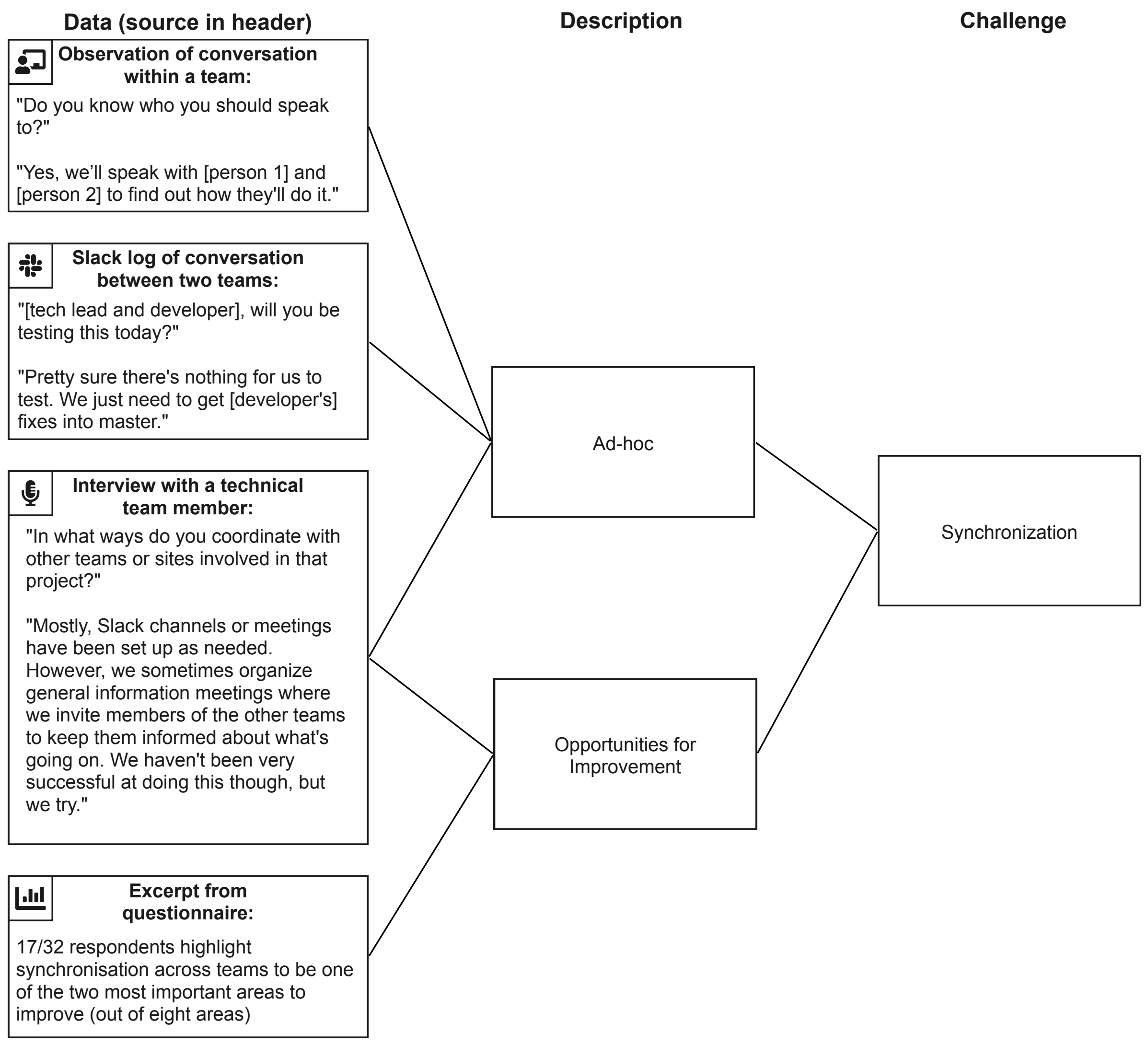
The choice of coordination mechanisms has proven to be context-depend (Šmite et al., 2010). Therefore, I chose a case study strategy to investigate my research question rather than another strategy, for instance, a survey, which most often simplifies the complexities of the real world (Oates, 2006).

However, I only examined what is occurring at Signicat in spring 2022. According to Dingsøyr et al. (2018a), coordination mechanisms change over time. Thus, a longitudinal case study would have been a more proper choice to observe potential changes over time and give me a deeper understanding of Signicat. However, a longitudinal case study is out of the scope of this master's thesis (see Section 1.4).

The thesis could have benefited from more observation hours outside formal arenas and ceremonies. However, due to the coronavirus pandemic, this was not feasible in the first part of the case study period. As I spent considerable time observing the agile ceremonies of the two teams, did I neither prioritize staying at Signicat's offices over some time during the last part of the case study period. Conducting a master's thesis involves some prioritizations and some of the ones I took include toning down ethnographic data collection procedures such as participative observations and long-term fieldwork. That said, my notetaking during the observations has been extensive, and the transcribed interview data covers some of the negatives of not observing the participants outside meeting rooms and agile ceremonies.

The findings of my master's thesis rely on data from only one case within a single organization with a restricted number of interviews (13 out of 110 agile team personnel, roughly) and questionnaire respondents (31 out of a possible 120, roughly). More cases, more organizations, and a more extensive data collection (despite the usage of four different data collection methods) may have identified additional or disconfirming findings and limited the significance of context.

The limitations of this case study include the potential issue with *external validity*. There is a potential bias that the questionnaire respondents might be more interested in the topic investigated than the average person (at Signicat) working with software development. The observed teams and interviewed personnel



Data (source in header)
Description
Challenge
Observation of conversation within a team:
"Do you know who you should speak to?"
"Yes, we'll speak with [person 1] and [person 2] to find out how they'll do it."
Slack log of conversation between two teams:
"[tech lead and developer], will you be testing this today?"
"Pretty sure there's nothing for us to test. We just need to get [developer's] fixes into master."
Interview with a technical team member:
"In what ways do you coordinate with other teams or sites involved in that project?"
"Mostly, Slack channels or meetings have been set up as needed. However, we sometimes organize general information meetings where we invite members of the other teams to keep them informed about what's going on. We haven't been very successful at doing this though, but we try."
Excerpt from questionnaire:
17/32 respondents highlight synchronisation across teams to be one of the two most important areas to improve (out of eight areas)
Ad-hoc
Opportunities for Improvement
Synchronization

Figure 3.2: Example of how data were analysed. Visualizing this way is inspired by Stray and Moe (2020)

were also only working from Trondheim and consisted mainly of Norwegians, which may influence their opinions on team autonomy and collaboration with other teams and parts of an organization.

Since my research is exploratory, *internal validity* does not apply as it is concerned only with explanatory or causal relationships (Yin 2009).

The construct validity of a case study concerns the identification of the correct operational measures for the concepts being studied (Yin, 2014). The main threat to the construct validity of this study was the choice of dimensions for the team autonomy model (4.5). I used the work of Moe et al. (2021) as a basis to decide on the four dimensions, and those choices relate to the accuracy of my interpretation of Moe et al. extensions of Hackman 's (1986) authority matrix, i.e., the types of decisions software engineering teams face. I suggested four statements to gather data on the four dimensions (see Section 2.2.2).

Regarding the interview data, a potential threat to validity is that I rely first on the interviewees' understanding of team synchronization, transparency, team autonomy, and inter-team coordination mechanisms, and then my interpretation of those terms when evaluating the findings. However, I have sought to ensure *reliability* through detailed descriptions of the data collection and data analysis and a thorough presentation of the data material in the results section (Section 4).

To improve the validity, I have utilized different types of *triangulation* (Oates, 2006). Triangulation is an approach to investigating a research problem with two or more perspectives. I have employed triangulation in the choice of methods, time, and space. Firstly, I used four methods for generating data: interviews, observations, questionnaires, and documents. Secondly, my observations of the two teams occurred at two different points. However, I consider the relatively short period between those two periods (3 weeks) not to be a significant factor in enhancing my findings' validity. Thirdly, I can argue that I presumably have conducted my study in two or more countries as I distributed the questionnaire to all sites of Signicat, including Scandinavian and Benelux countries, as well as Portugal, Lithuania, and Romania. That said, the questionnaire was anonymous, and it might be the case that just one site of Signicat responded to the questionnaire. I also consider my mixed-method approach, utilizing both qualitative and quantitative data analysis techniques, a strength of the study. Nevertheless, I believe the study would have benefited from investigator triangulation. A fellow researcher and I could have discussed and compared findings, and we could have set a broader and deeper theoretical foundation. Additionally, a more comprehensive data collection and a more thorough discussion in Chapter 5 would have been likely outcomes of such a triangulation.

4 Results

In this case study, I investigate how to balance alignment and team autonomy in large-scale agile software development and how inter-team coordination mechanisms affect this balance. From my analysis, I identified 12 inter-team coordination mechanisms. The analysis of the identified inter-team coordination mechanisms is structured according to the inter-team coordination taxonomy proposed by Berntzen et al. (2022). The taxonomy includes three main categories, namely meetings, roles, and tools and artifacts, which are further divided into six sub-categories: scheduled and unscheduled meetings, individual and team roles, and tangible and intangible tools and artifacts. I present the taxonomy in more detail in the theory chapter (Chapter 2). Additionally, all mechanisms have been examined according to the TOPS framework (Berntzen et al., 2022) to identify each mechanism's underlying characteristics (technical, organizational, physical, and/or social). As described in the theory chapter (Chapter 2), inter-team coordination is necessary to manage inter-team dependencies, and I therefore also use the dependency taxonomy of Strode (2016) to analyze which dependencies each of the 12 inter-team coordination mechanism manage.

Table 4.2 provides an overview of all 12 inter-team coordination mechanisms and their TOPS characteristics and a brief description of which dependencies they manage. As the table illustrates, I have merged some mechanisms to count as one (table design by Berntzen et al. 2022, page 11).

However, before diving into the details of the identified inter-team coordination mechanisms, I follow up the introductory presentation of Signicat in Chapter 3 by providing more details on the case context and by presenting an analysis of the questionnaire data I collected. The first provides in-depth information about Signicat and the context in which the agile teams operated, particularly the two teams I observed in-depth, and the latter gives an overview of the agile teams' perception of certain aspects regarding team autonomy and inter-team coordination at Signicat.

4.1 Case Description

Signicat is recognized as a leading provider of digital identity solutions in Europe, with numerous companies utilizing their services, most being customers with high security requirements in regulated industries. Their product portfolio includes solutions for identity verification, authentication, and electronic signing, and their platforms offer connections to more than 30 electronic identification 's (eID) companies can use to connect to their customers to comply with security requirements and regulations (e.g., Norwegian BankID, Swedish BankID, and MitID name but a few). The complete list of available eID 's can be found at Signicat's developer portal[1]

Signicat was established in Norway in 2007 and has since expanded its business significantly. Over the past few years, they have gone through several company acquisitions and are now employing more than 400 people across several European countries spanning regions such as the Nordics, Baltics, Benelux, DACHs, and Balkans. However, their main technology sites were Oslo, Bergen, Trondheim, Vilnius, Rotterdam, Estoril/Lisboa, and Bucharest. The two teams I observed in-depth (Team Signature and Team Nordic eID) were located in Trondheim (see Figure 4.1).

Signicat was and is currently transforming from several technology platforms (also called "stacks," and more specifically green stack, blue stack, and orange stack) to a unified one (see Figure 4.2). Their main argument for initiating this transformation process is that their fragmented technology platforms of today sooner or later will hinder further growth and increasingly complicate the design of services relying on cross-stack features. Their previous strategy to inter-connect technology stacks started to raise several issues: reduced agility in creating cross-stack service combinations; confused customers and staff regarding forward integration path; and potential cost synergies suffering from duplicated services and a

[1]https://developer.signicat.com/enterprise/docs/authentication/apis

fragmented tech platform. To mitigate these issues, they initialized a transformation towards a unified platform for the agile teams and their customers. Additionally, they considered a future common and coherent developer experience to make them more flexible as a global organization.

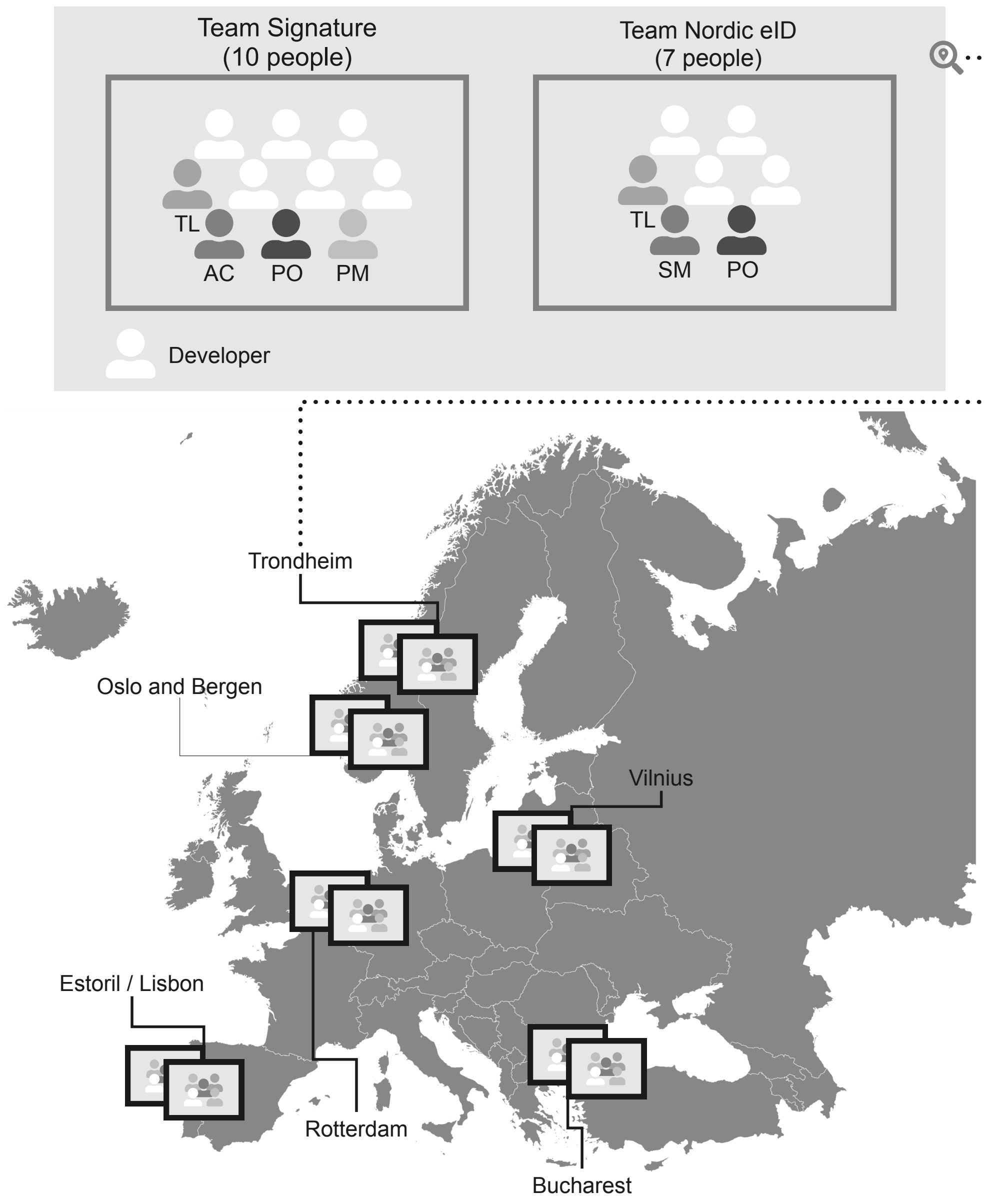


Figure 4.1: Main technology sites of Signicat. Team Signature (10 people) and team Nordic eID (7 people) were located primarily in Trondheim. Team Nordic eID relied upon country product managers (CPMs), albeit the CPMs worked across all teams, whereas team Signature had a product manager (PM) within the team. Additional roles were developer, agile coach (AC), scrum master (SM), product owner (PO) and tech lead

In parallel, Signicat was and is currently undergoing an organizational transformation. Recently, they have implemented a tribe structure to better connect the technology and product environments and more successfully integrate the sales units (see Figure 4.6). The agile teams operated within this tribe structure and were clustered in tribes according to product areas. During the case study period (spring 2022), there were 14 agile development teams involving around 110 people, of whom roughly 15 were tribe personnel, distributed among seven tribes. Many teams in Signicat were co-located, but also some of the teams were internationally distributed. However, Signicat aimed not to have teams covering more than two or three different sites. The agile teams came in two types, product teams and platform teams, each with its set of designated roles. Whereas the product teams developed services and solutions used externally (customers) and hence had resources from the product department (product manager (PM) and product owner (PO)), the platform teams mainly developed and facilitated internal solutions in which the product teams relied upon, and hence consisted only of personnel from the technology department (no PM, and platform owner instead of PO). In Table 4.1, I present an overview of the different team roles in the agile teams in Signicat. I omit user experience (UX), tech writer, and quality assurance (QA) personnel from the table, although most teams were allocated these resources. I also omit the tribe lead and tribe architect roles from the table as these were tribe roles, though closely connected to the agile teams. Instead, these five roles are presented as inter-team coordination roles in Section 4.4 since they ensured alignment between teams.

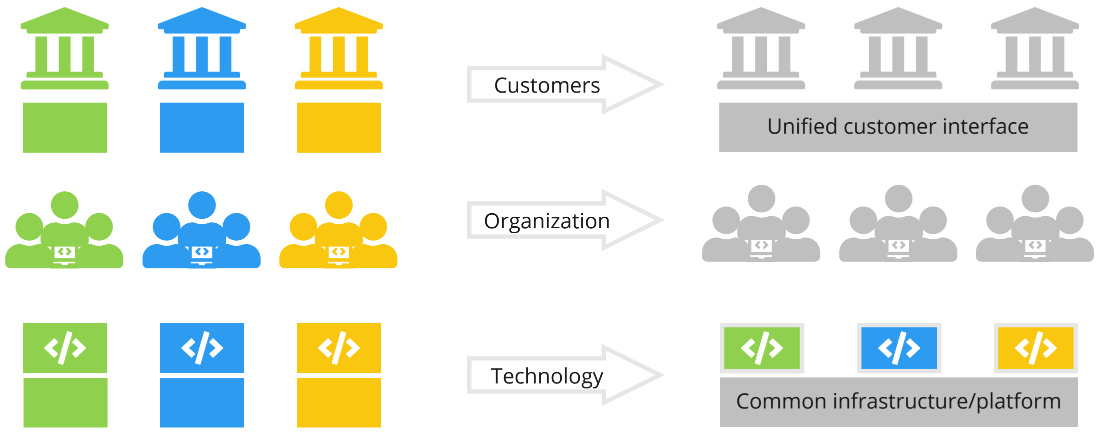


Figure 4.2: Signicat's transformation from inter-connected technology stacks to a unified platform. Until recently, the agile teams and Signicat's customers relied on different technology stacks and platforms. The figure design is a minor redesign of Signicat's illustration of the transformation

Together with the tribe structure, seven communities of practice (named guilds at Signicat) formed the two major elements within which the agile teams operated. Figure 4.6 provides an illustration of this context. These elements may invoke associations with how Spotify scaled agile at the beginning of the previous decade (see Section 2.5). However, Signicat has not implemented *the* Spotify Model but found their model and way of working with tribes. The tribe structure clustered the agile teams operating in the same product area. The guilds provided a foundation for aligning a common direction and best practices on certain matters across the agile teams and tribes, just as the Spotify Model recommends. However, as described in Section 1.4, this master's thesis is written with a software engineering lens, and I therefore simplify some organizational aspects: the technology department at Signicat is a line organization, but I am only focusing on the agile team personnel (and tribe leads and architects) since the line managers were not accountable for daily planning or follow-up on team activities; I also exclude departments such as sales and marketing when outlining the organizational structure; and I am as well omitting the role of managers and C-executives. By providing an overview of the tribe structure and outlining how the agile teams interconnected through the guilds, I argue that I have brought the most relevant details regarding organizational structure to the table from a software engineering lens.

In the following, I present the two agile teams (Team Signature and Team Nordic eID) I observed in-depth during the spring in more detail before outlining the analyzed questionnaire data.

Team Signature

Team Signature was co-located in Trondheim and operated as the only team within the Signature tribe (see Figure 4.6). The tribe lead was however located in Oslo. The team developed solutions for electronic signatures and seals across Europe, which often relied upon eID services, for instance, integrated by Team Nordic eID (presented in Section 4.1). Whereas an electronic signature is data in an electronic form used by a neutral signatory to sign, e.g., a document, an electronic seal is data in an electronic form used by a legal person to ensure the origin and integrity of the data of, for instance, a document[2]. The signature services were web-based and could be integrated into websites, applications, or business systems using Signicats REST APIs or SDKs[3]. Customers of Signicat could also let their customers sign contracts, declarations, forms, or other documents by using *digital signatures*, a variant of electronic signatures based on PKI, Public Key Infrastructure (for the sake of simplicity: a mathematical and cryptographic concept).

Before my case study period started, team Signature had replaced their Scrum with a Kanban-based process. As described in Section 2.1.2, roles are less central in Kanban compared to Scrum, but Team Signature kept following the Signicat guidelines for team composition and roles (presented in Table 4.1), though their agile coach was interim. They were as well running several agile ceremonies on the team level during the case study period, including:

- Backlog refinement (70 minutes bi-weekly. No developers attended, typically. Further presented in Section 4.3 since tribe personnel had an active role during this meeting)
- Planning meeting (60 minutes bi-weekly)
- Readiness meeting (30 minutes weekly. Certain developers could sometimes attend depending on the topic of discussion)
- Demonstration meeting (60 minutes monthly)
- Bi-daily stand-up meeting (20 minutes midday, alternating between three days a week and two days a week)
- Retrospective meeting (60 minutes monthly)

Since Kanban emphasize continuous flow rather than fixed-length sprints (see Table 2.1), there may be a little more challenging to track key performance indicators (KPIs) reasonably compared to Scrum teams, but several interviewees were determined that they do what "*feels best for the team to create the best product*" and that they anyways "*had tried both methods, and agreed that their Kanban was more agile and worked better than the Scrum.*"

Team Nordic eID

Team Nordic eID was responsible for the integration of Nordic eID's, for instance the Norwegian BankID and the Swedish BankID in the Authentication tribe (see Figure 4.6). Like team Signature, they were co-located in Trondheim and had adopted team roles in line with the Signicat guidelines for team composition and roles (presented in Table 4.1). Albeit, their tribe lead and tribe architect were operating from the Rotterdam office. The electronic eID services allow customers of Signicat to verify/authenticate the identity of users who sign up and log into a service with national or public eID 's. Many of the signature solutions of Team Signature were dependent on such authentication. That said, the Nordic eID team and the Signature team did not necessarily work closely together, yet their services certainly related to each other. Signicat also offered authentication services based on biometric scanning, which are compliant with an even higher level of assurance than eID 's. However, this product area belonged to another team within another tribe (Mobile ID tribe).

At the team level, Team Nordic eID had adopted the Scrum framework. They ran iterations of three weeks, which differed from the other Scrum teams at Signicat, which ran sprints of two weeks. However, team Nordic eID had previously run iterations of two weeks as well, but as they experienced two weeks to be "simply too short (for us) to produce anything of notable value," they argued why their team and the eID product would benefit from increasing the iteration length by one week. By running two weeks, they too often experienced that *"a new round of ceremonies simply came too soon"* [developer], and that there

[2]https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eSignature+FAQ
[3]https://developer.signicat.com/enterprise/docs/electronic-signatures/introduction

were *"too little time for the developers to actually develop software"* [developer]. However, the three-week cycle did not fit very well with the two-week cadence of the other teams, for instance, concerning reporting key performance indicator (KPI) trendlines, so transitioning to three weeks had to be argued for.

Team Nordic eID used a sort of Kanban boards to manage their workloads, including columns such as "sprint backlog" (aka todo's), "in progress," "on hold," "in review," "ready for release" and "done" (from left to right). The board was split into rows containing sub-project tasks (e.g., tasks on different eID 's) within the team.

The team had adopted all the artifacts and events of the Scrum framework (see Figure 2.1). Additionally, the PO had a prioritization meeting with CPMs every third week. In total, these were the regular meetings of the team:

- Backlog refinement with CPMs (30 minutes every third week. Only the POfrom the team. Further presented in Section 4.3 since different geographical market areas coordinated during this meeting)
- Backlog refinement (90 minutes every third week)
- Planning meeting (3 hours every third week)
- Sprint review (2 hours every third week)
- Sprint retrospective (90 minutes every third week)
- Daily stand-up meeting (15 minutes around midday five days a week)

Role	**Description**
Product Owner (PO) or Platform Owner	Responsible for the backlog and correct prioritization, and for communicating requirements (both functional and non-functional) to the rest of the agile team. Some PO 's represented their team in the tech excellence guild. PO for product teams, platform owner for platform teams.
Product Manager	Responsible for the "why, when, and what" related to products. Worked closely with the PO such that the PO was able to define work for the agile team.
Scrum Master (SM) or Agile Coach	Accountable for driving the agile ceremonies of the agile team. Represented the team in the tech excellence guild. SM for Scrum teams, agile coach for Kanban teams.
Tech Lead	Responsible for setting and steering the team's technical direction. Aligned with the tribe architect on key technical decisions and overall design, and ensured compliance with decisions/guidelines by the architecture guild
Developers	4-7 seven developers were allocated to a team (a mixture of junior and senior developers). Some also undertook responsibilities in either the DevOps or security guild

Table 4.1: Team roles at Signicat

4.2 Development of a Team Autonomy Model

In this section, I analyze how agile team personnel perceived the current state on certain topics regarding team autonomy and inter-team coordination at Signicat. The analysis is based on collected data from 31 anonymous questionnaire responses. Hence I do not know the respondents' location or position within the agile teams. The questionnaire contained eight statements, four on team autonomy (Figure 4.3) and four on inter-team coordination (Figure 4.4). The respondents specified their level of agreement to a statement on a five-level Likert scale: strongly agree; agree; neither agree nor disagree; disagree; and strongly disagree. Additionally, the respondents were allowed to pick one, two, or three areas (out of the eight areas covered in the eight preceding statements of the questionnaire) they wanted to highlight as the most important to improve at Signicat. Among the 31 respondents, hence 93 possible selections for improvement, 77 picks were collected, meaning that not all respondents considered that three areas of those covered in the questionnaire called for improvement. The histograms in Figure 4.3 shows the distribution of answers on the four team autonomy statements, and the histograms in Figure 4.4 shows the distribution on the four inter-team coordination statements. The topic each statement/histogram covers is stated in the bold title on top of each histogram. The red horizontal line within each histogram shows the number of times the respective topic was chosen as one of the three most important areas to improve.

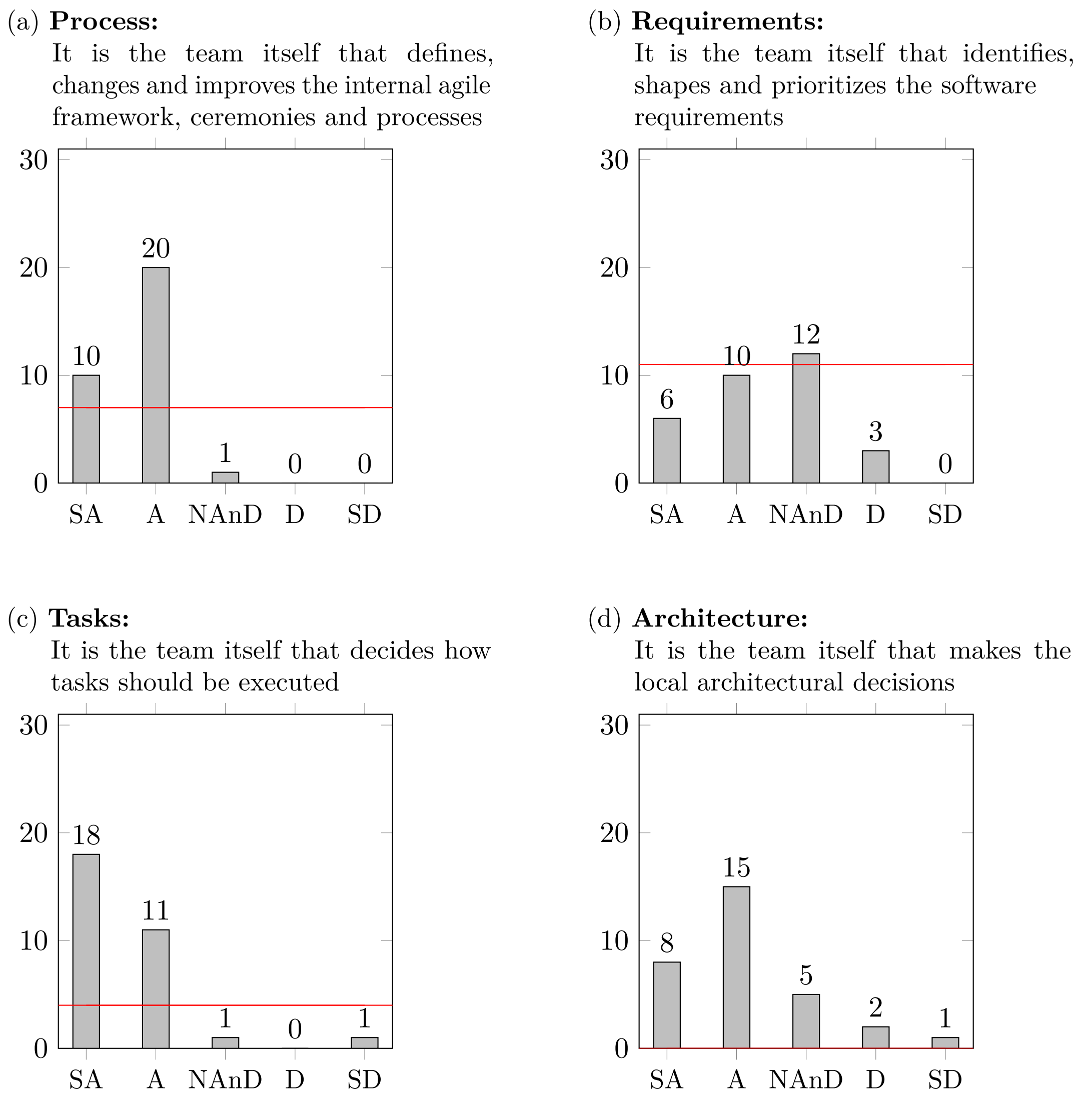


Figure 4.3: Distribution of the responses on the four statements regarding team autonomy in the questionnaire.
SA=Strongly Agree, A=Agree, NAnD=Neither Agree nor Disagree, D=Disagree, SD=Strongly Disagree
—Picked by respondents as one of the three most important areas to improve

Figure 4.3 shows that a vast majority of the respondents either agreed or strongly agreed that the agile teams self-decided their internal work processes (30/31, see Figure 4.3a) and how tasks should be executed (29/31, see Figure 4.3c), while there were less agreement on the level of autonomy on architectural decisions and software requirements (see Figure 4.3d and 4.3b, respectively). However, none of the

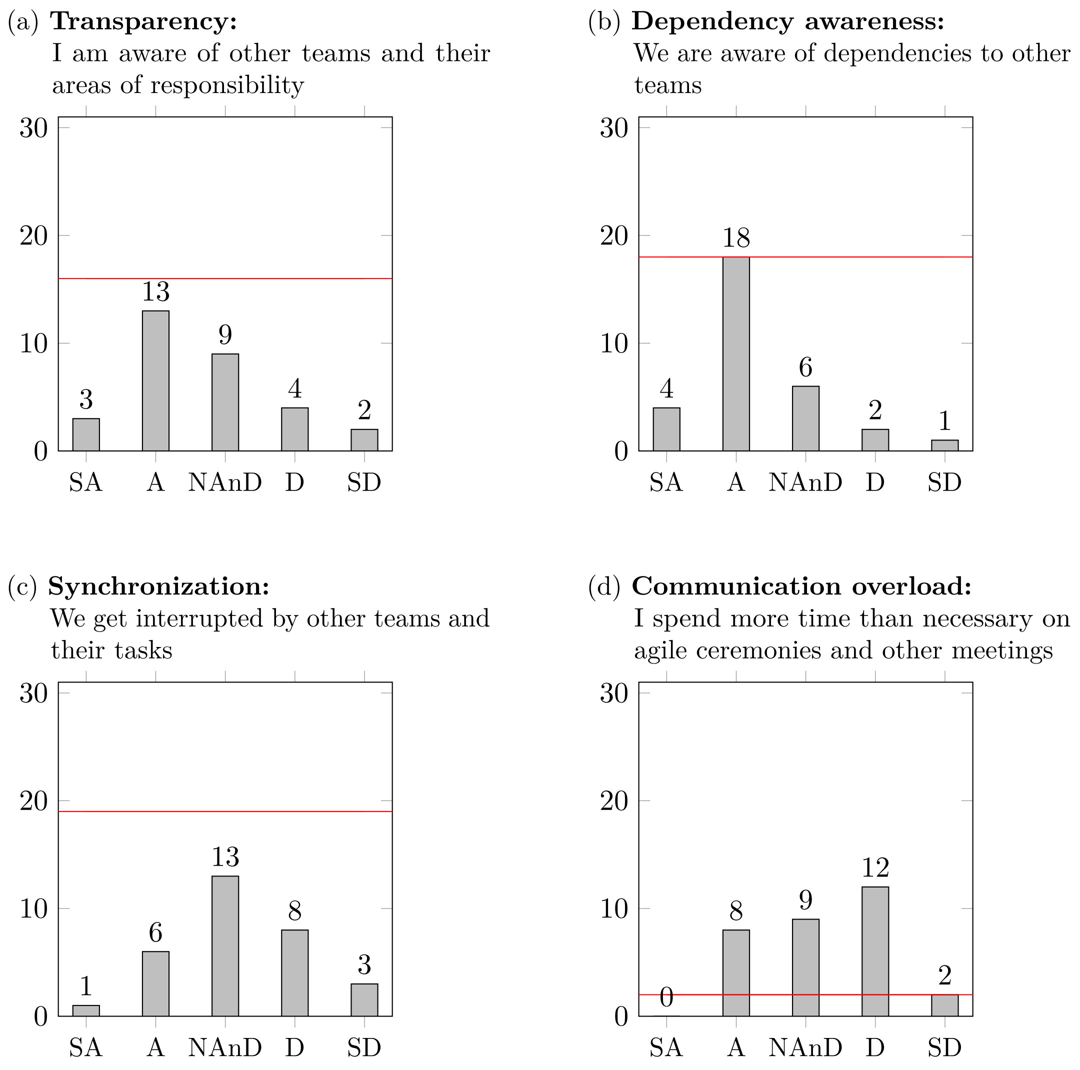


Figure 4.4: Distribution of the responses on the four statements regarding inter-team coordination in the questionnaire.
SA=Strongly Agree, A=Agree, NAnD=Neither Agree nor Disagree, D=Disagree, SD=Strongly Disagree
—Picked by respondents as one of the three most important areas to improve

respondents picked architecture autonomy as one of the three most important areas to improve, whereas 11/31 picked the level of autonomy on software requirements as one (indicated by the red horizontal line).

Upon considering inter-team coordination, the respondents' answers were a little more distributed (see Figure 4.4). 12/31 disagreed that they spent too much time on agile ceremonies and other meetings, 8/31 agreed, and nine neither agreed nor disagreed. A similar distribution of answers was seen when considering if teams were interrupted by each other (see Figure 4.4c). The respondents were more in agreement regarding dependency awareness (18/31 agreed that they were aware of dependencies to other teams, see Figure 4.4b).

I identified a greater call for inter-team coordination improvements than team autonomy. The four areas regarding inter-team coordination were picked 55 times as areas to improve at Signicat, whereas the four aspects regarding team autonomy were picked 22 times. With a total of 77 picks, 16 of the possible 93 were not used (each respondent could pick three areas for improvement). Transparency (Figure 4.4a), dependency awareness (Figure 4.4b), and synchronization (Figure 4.4c) were by far picked most times as one of the three most important areas to improve with 18/31, 17/31, and 19/31, respectively. Contrary, communication overload (Figure 4.4d) was picked only two times by the 31 respondents (with 93 possible picks) as an area to improve.

To develop the team autonomy model, I derive mean values on each of the four team autonomy statements (for more details, see Section 3.3). The model is displayed in Figure 4.5. Each corner in the diagram represents one dimension. On the statement regarding team autonomy on process, ten strongly agreed, 20 agreed, one neither agreed nor disagreed, and none disagreed or strongly disagreed. Hence the mean on the process dimension is calculated to be 4.3. The other means were 3.6 on requirements, 4.5 on tasks, and 3.9 on architecture.

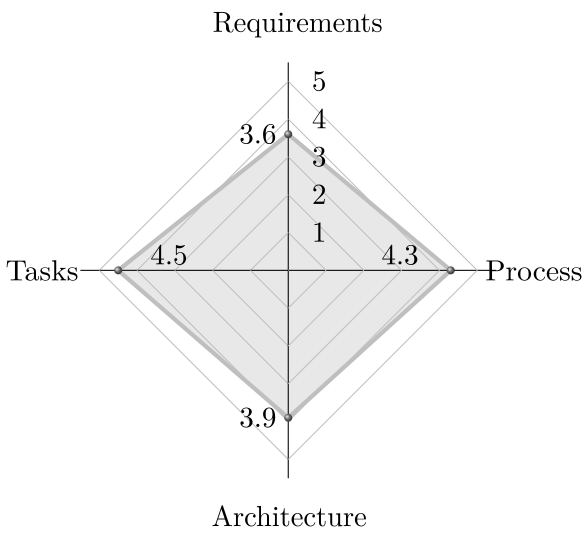


Figure 4.5: The level of team autonomy at Signicat on four dimensions

4.3 Inter-team Coordination Meetings

Several inter-team coordination meetings were held at Signicat. Table 4.2 presents all the seven types of meetings highlighting their TOPS characteristics and which type of dependencies they managed. I include both scheduled and unscheduled meetings. Some of the scheduled meetings were customized agile ceremonies such as the catwalks (company-wide internal demonstrations of any sort), the common retrospectives (mainly an arena for reporting KPI trendlines), the different types of refinement meetings, and the yearly company-wide kick-off meeting serving several purposes. Other scheduled meetings included weekly, bi-weekly, or monthly CoP meetings (called guild meetings at Signicat) and tribe lead meetings. In addition, numerous ad-hoc meetings occurred across teams and tribes, both digitally/remotely and on-site. All meetings managed knowledge dependencies, and almost all were related to technical product progress. However, they did serve different purposes. To illustrate, I describe the seven meetings and their characteristics in more detail.

Coordination Mechanism	T	O	P	S	Description
					MEETINGS (n=7)
Catwalks	☒	☐	☒	☒	A monthly demo for all employees. An arena for the development teams to showcase their work (the latest features and/or ideas), and hence an arena for managing knowledge dependencies. Conducted as a hybrid meeting.
Common retro-spectives	☐	☒	☒	☒	A monthly arena for sharing of agile best practises, key performance indicator, and key actions across teams. As such, process dependencies as well as knowledge dependencies were managed. Conducted as a hybrid meeting.
Guild meetings*	☒	☒	☒	☒	Team and/or tribe representatives met either weekly, biweekly or monthly to align and synchronize direction and best practices within the specific domain of the guild (referred to as CoP in the literature), which in total managed knowledge, process and resource dependencies. Conducted as a hybrid meeting.
Refinement Meetings (A)	☒	☐	☒	☒	POand CPMs met every third week to discuss and assess the product backlog of the team, and to do an overall prioritization of the upcoming sprint. The involvement of CPMs ensured the team stayed up to date on the status of different geographical market areas. Primarily focusing on product-related aspects. Managed process dependencies.
Refinement meetings (B)	☒	☐	☒	☒	agile coach, PM, tribe lead and tribe architect met bi-weekly (digitally) to refine and prioritize the team's product backlog. The focus was on product and technical requirements, and thus, this meeting managed knowledge and process dependencies.
Kick-off Meeting	☐	☒	☒	☒	Yearly full-day socializing conference bringing clarity to all employees regarding the firms' mission, its overall product vision and future direction, and the key targets for the upcoming year, as well as showing recently launched products. Managed knowledge and process dependencies. Conducted as a hybrid meeting.
Unscheduled Meetings	☒	☐	☒	☒	Cross-team ad-hoc meetings/chats held when needed, managing knowledge and resource dependencies. Conducted either physically (over desk or inside a meeting room) or digitally (by chat or by audio/video).
					ROLES (n=3)
Guilds*	☒	☒	☒	☒	The seven guilds (Architecture, DevOps, QA, Security, Tech Excellence, TechDoc and UX) aligned best practices across tribes. In total, process and resource dependencies were managed. Referred to as CoP's in the literature.
Platform teams*	☒	☐	☒	☒	Platform teams managed technical resource dependencies by providing product teams common platform services.
Tribe Leads	☐	☒	☐	☒	Drove strategy and operations within a tribe's product area and ensured resource utilization. Managed knowledge, resource and process dependencies. Compare to the PM at the study of Entur by Berntzen et al. (2022).
					TOOLS AND ARTIFACTS (n=2)
Communication Tools*	☒	☐	☐	☒	Tools (e.g. Slack and Zoom) managed resource and knowledge dependencies by facilitating communication and information sharing. Slack, Zoom and Outlook are three examples.
Documentation Tools*	☒	☒	☐	☒	Tools (e.g. Confluence and Jira) managed resource and knowledge dependencies by facilitating information sharing and supporting development processes. This included artifacts such as OKRs, knowledge bases, task boards, roadmaps, burdown charts, product backlogs, and overview of the organisational structure.

Notes: An asterisk (*) indicates that similar mechanisms were collapsed into one. TOPS characteristics of a mechanisms are illustrated with ☒. PO = Product Owner. PM = Product Manager. CPM = Country Product Manager. DevOps = Development and Operations. QA = Quality Assurance. TechDoc = Technical Documentation. UX = User Experience. OKR = Objective and Key Results. The types of dependencies, following Strode (2016), are described in Chapter 2

Table 4.2: Inter-team coordination mechanisms per taxonomy category and TOPS characteristic (Berntzen et al., 2022) at Signicat. Same table design as Berntzen et al.

Catwalks. The catwalks were primarily an arena for teams to *"showcase the latest work to other employees at Signicat"* [PO], and hence, managing knowledge dependencies across all teams. The meeting occurred every other week for one hour, and all employees across all sites were invited and free to attend. Because it spanned all locations of Signicat, the meeting was held as a hybrid meeting: typically, a meeting room at each site was booked, thus having a physical and social characteristic, and each site was then connected through a Zoom[4] meeting session. The number of participants varied, but during the spring, an average of 135 participated in each catwalk (the average calculation is based on the number of accepted invitations of seven catwalks during spring 2022).

Common retrospectives. Every month, the scrum master (SM)/agile coaches of each team met for one hour to share key performance indicators, key action points, and agile best practices and thus managed process and knowledge dependencies. However, the arena seemed to be focusing more on KPI 's than aligning of best practices: *"I wish the arena were used more frequently to address collaboration problems between teams, but that said, it is a large arena, so such issues might be better addressed within an arena with fewer teams"*[SM]. The KPI reporting of each team includes completion rate (delivered tasks divided by the amount planned) and productivity (delivered issues divided by the number of developers). The meeting leader aggregated the KPIs of each team to calculate an average of the KPI 's across teams for graphical representation to discover emerging company trends. As the meeting involved teams from different sites, the meeting was conducted digitally. However, teams at the same site typically sat together in a booked meeting room. Thus physical and social characteristics are illustrated as well.

Guild meetings. The guild meetings allowed team members of different teams within different tribes to meet, either weekly, biweekly, or monthly (depending on which guild they belonged to), to align and synchronize direction and best practices within the specific domain of the guild. The length of the meetings varied from 30 minutes to 90 minutes. The meetings served as an arena for technical coordination, thus managed knowledge, process, and resource dependencies. As the guild meetings connected personnel across teams and tribes, the meetings also had a social characteristic. The different guilds are described in more detail as inter-team coordination *roles* in Section 4.4.

Refinement meeting (Team Nordic eID). Every third week prior to the start of a proceeding sprint, the PO of team Nordic eID met with country product managers (CPMs) (typically three or four) for one hour to discuss and assess the team's product backlog. The PO updated the CPMs on the current product roadmap of the team, primarily focusing on the shorter perspective. Therefore, these meetings were characterized primarily as a technical inter-team coordination mechanism. The CPMs shared insights on what functionality is needed to stay competitive within their respective areas and issues regarding already developed features. As these meetings focused on product-related requirements, they managed knowledge dependencies. Involving CPMs in the overall prioritization allowed the PO and hence the team to stay up to date on the needs of different geographical market areas. The meetings also had an organizational characteristic, as discussions on the prioritization process across teams and tribes occurred, albeit this was not the intention of these meetings. Since the CPMs and the PO were at different locations, these meetings were held digitally. The length of these meetings decreased from one hour to 30 minutes during the spring as the participants experienced the meetings being more effective.

Refinement meeting (Team Signature). Every second week before an upcoming two-week cycle, the PO, agile coach, and PM of team Signature met for 70 minutes to refine and prioritize the team's product backlog. Usually, the tribe lead and tribe architect of the Signature tribe did also attend these bi-weekly meetings. The rest of the team (i.e., the developers) was not intended to attend these meetings except in some very specific cases where their oral feedback was necessary. However, this did not happen during my observations. Primarily, these meetings focused on feature- or product-related progress, thus serving as a coordination mechanism for managing knowledge and process dependencies. The participation

[4]https://explore.zoom.us/en/about/

of the tribe personnel ensured the tasks on the top of the backlog were the correct ones to include in the upcoming two-week iteration.

The outcome of the refinement sessions could either be a re-prioritization of the (top of the) backlog, creation of new issues, or closing issues. A dedicated Jira refinement board was used for these meetings. As the tribe lead was located in Oslo, and the team and tribe architect sat in Trondheim, these meetings were held digitally/as a hybrid.

Kick-off meeting. Each January, a kick-off meeting is arranged for all Signicat employees. The meeting is conducted as a hybrid meeting: different sites, sitting together (physical characteristic), were connected through a joint video conference session. It is a full-day socializing conference presenting Signicat's mission, the recently launched products, the company's overall product vision, future direction, and the key targets for the upcoming year, thus an arena for managing knowledge dependencies. The meeting also facilitated informal socializing and served as an arena for creating and sharing a positive attitude, thereby bringing a social characteristic. That said, I did not attend this meeting myself. The reason is that the meeting was held in the first part of January, whereas I began my observations in the last days of January. However, I have read through the presented material, watched some of the recorded parts of the 2022 meeting, and collected insights during my interviews.

Unscheduled meetings. The unscheduled meetings were the most regular inter-team coordination meeting to manage knowledge, resource, or process dependencies. The length of ad-hoc meetings differed greatly, and they occurred frequently to resolve inter-team dependencies: *"Often, it [a dependency] can be solved just by talking to each other, instead of involving architects or whoever"* [Developer]. Such conversations enabled quick coordination and decision-making and were facilitated by, but not limited by, co-location and open landscape. However, physical proximity and co-location were valued by many: *"I believe that having lunch and coffee together is an important part of bonding collegial relationships. My experience is that spending some time together at least a few days a week enhances productivity, as well as well-being and motivation among the team, and these factors may contribute to the overall success of a project"*[PM]. Slack[5] figured as a substitute for physical ad-hoc discussions when necessary and was used extensively as a platform for information sharing across teams and sites. However, the threshold of initiating a digital chat and/or meeting seemed to be a bit higher than the case for co-located employees to initiate an ad-hoc conversation at the office. One of the reasons much inter-team coordination took place ad-hoc was that *"all teams do not have established channels of communication between one another, such as a Scrum-of-Scrum or meetings of a similar format."* [Developer/SM].

4.4 Inter-team Coordination Roles

Like Berntzen et al. (2022), I consider roles external to the teams as the inter-team coordination roles. This includes both individual roles and team roles. Table 4.2 presents the three types of identified roles and their TOPS characteristics in more detail. The different types of roles served different purposes: whereas the architect guild had a stake in technical and structural discussions and thus managed resource, business process, and knowledge dependencies, the platform teams provided the teams with a common environment managing technical resource dependencies; and whereas some guilds devoted resources (personnel) to the agile teams, some guilds were rather composed of agile team members (see Figure 4.6). Seven guilds were operating across teams and tribes at Signicat, aiming to align and synchronize direction and best practices in certain domains. The guilds were composed of employees with skills, interests, or stakes in the discipline under consideration. In the following, I describe the seven guilds in more detail.

[5]https://slack.com/about

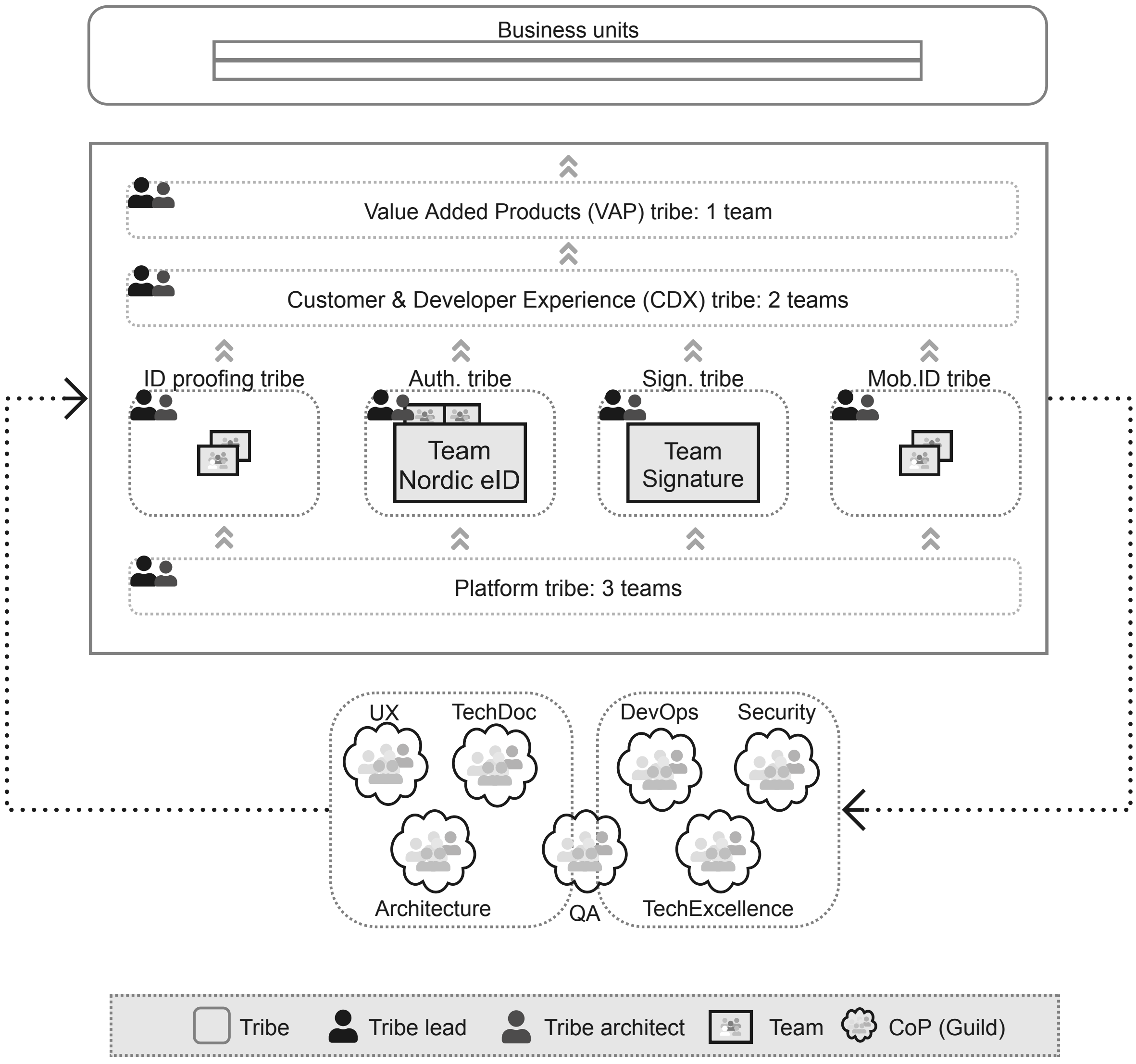


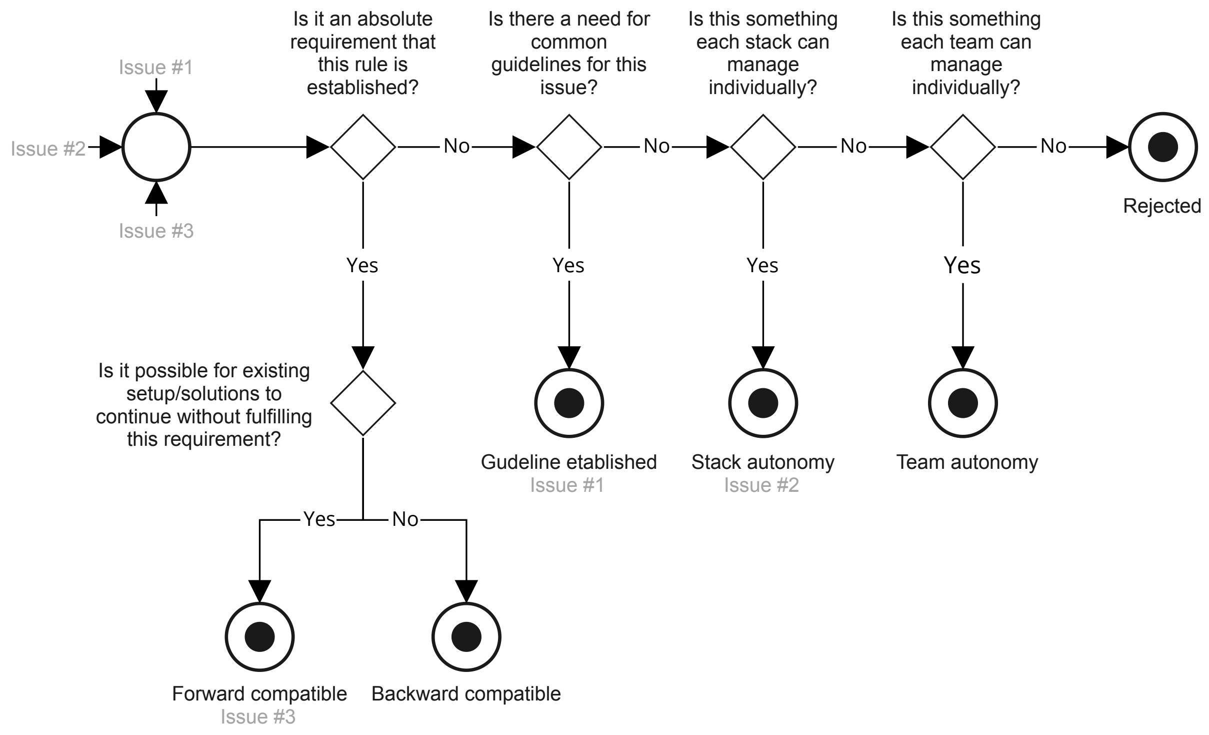
Figure 4.6: The work context of the agile teams at Signicat. The figure shows the seven tribes, the seven guilds, and a sketch of the business units which sell products from all tribes. Each tribe is led by a tribe lead and designated a tribe architect. The three platform teams manage the software foundation upon which team Nordic eID, team Signature, and the other teams above the platform tribe relied. The CDX tribe and VAP tribe use and integrate services from all tribes underneath such that Signicat can offer combinations of products "out-of-the-box" to customers (i.e., tying the services/features/products of the other tribes together). The seven guilds either provide the agile teams/tribes with resources (personnel, e.g., a UXer) or are composed of personnel from the agile teams (e.g., a developer figuring as a "security champion")

Architecture Guild

The architecture guild was a group whose purpose was to coordinate architecture, technology standards, and associated best practices across tribes, ensuring the company moved in a coherent technological direction. The group consisted of the seven tribe architects and the security guild lead (lead of another guild at Signicat). Other people with a stake or knowledge could also be invited to the guild for some time. The lead architect led the group in Signicat, and they met every other week for one and a half hours to discuss and elaborate on relevant topics. The architect guild also operated a Slack channel open to everyone in the organization, and ad-hoc meetings occurred regularly.

Anyone in the organization could suggest issues to be considered by the architecture guild by creating issues in the guild's dedicated Jira board. The architecture guild assessed every issue using a decision flow diagram illustrated in Figure 4.7. The flow diagram also shows the assessment of three real-case architecture issues at Signicat with differing outcomes:

- Issue #1 (REST guidelines) is defined as a guideline on the autonomy scale, meaning new development of REST APIs is recommended to follow the defined guidelines. Already existing REST APIs may be upgraded to follow the guidelines. The audience for the guidelines is the whole company.
- Issue #2 (Choice of containerization technologies) is defined as a forward compatible standard. Meaning k8s [Kubernetes] shall be the choice for container orchestration as long as it exists as a managed service. This is applicable for future containers. The standard applies to the whole company.
- Issue #3 (Automation of non-containerized workloads) is defined as stack autonomy, as each stack should agree on the choice for automation in non-containerized applications. Each stack can choose that this is team autonomy.



Issue #1
Issue #2
Issue #3
Is it an absolute requirement that this rule is established?
Is there a need for common guidelines for this issue?
Is this something each stack can manage individually?
Is this something each team can manage individually?
No
Yes
Rejected
Is it possible for existing setup/solutions to continue without fulfilling this requirement?
Gudeline etablished
Issue #1
Stack autonomy
Issue #2
Team autonomy
Forward compatible
Issue #3
Backward compatible

Figure 4.7: Decision flow diagram for assessing architecture issues used by the architect guild at Signicat. Stack autonomy refers to autonomy within each technology platform (see Figure 4.2). Example issues #1, #2, and #3 are presented in the text

Autonomy reflects that no decision is made, and the team (or teams developing features in a certain technology stack) can carry out their mission with zero to minimal involvement from external parties. Thus autonomy prevails.

A *guideline* can either be best practice statements or strong recommendations, where the latter indicates an important guideline. A *standard* is a requirement applying to the whole organization, either valid from the time of establishment (forward compatible) or requiring existing projects to adopt the new standard (backward compatible). This is the level of most external control. However, the architecture group strives for team autonomy, being highly appreciated by the tech leads and developers: *"It is important for the developers' well-being that they do not just get dictated"* [Tech Lead], and most of the daily local architectural choices, such as *"code structure and which classes to create and things like that"* were *"mostly discussed and clarified at team level"* [Developer].

DevOps Guild

The development and operations (DevOps) guild was a community providing its participants an arena for asking DevOps-related questions, presenting and sharing DevOps best practices, and deciding on company-wide DevOps practices that should be standardized. This includes decisions and training on tooling, automation, deployment, monitoring and alerting, and reacting and resolving incidents. The guild consisted of 20 people, including developers and POs across the company and the seven tribe architects. However, the guild emphasized transparency, and anyone could ask to participate in the guild's bi-weekly meetings and suggest solutions and ideas to the guild. The guild was led by a guild lead (the interim agile coach of Team Signature) responsible for facilitating the group's activities and ensuring sufficient DevOps training exists in all tribes and teams. The guild operated its own open-access space in Confluence, which showed the guild's current work items, a decision log showing the history of the decision made and their outcomes, relevant DevOps literature, and meeting notes.

QA Guild

The quality assurance (QA) guild's mission was to agree upon and share best practices on tools and processes for ensuring software quality across Signicat. Each team had a QA representative in the guild that acted as the test and QA "champion" for the team, and the guild consisted of 18 people led by a QA guild lead. However, each agile team was collectively responsible for testing and quality, but the QA champion facilitated with knowledge about tools, good practices, and guidance. The QAs seemed to be successfully embedded within their respective teams, even if they *"work more horizontally and among the firm's different teams and sites"* [Developer].

The guild arranged frequent status meetings (typically once a week), workshops, and training when needed to maintain QA domain knowledge. The purpose of the status meetings was to share and inform the rest of the guild about upcoming news from the different development teams and discuss improvements, activities, and releases done during the week.

Security Guild

Each team had a developer designated security "champion" to build and share security knowledge across teams. The champion was part of the security guild, and *"once a month, we meet to discuss topics related to security"*[Developer]. The guild's duty was to help all teams establish a secure development process compliant with best practices, standards, and policies. The champions introduced and supported the agile teams with relevant security tools (e.g., how to perform static code analysis): *"We want to use the same technologies across the firm and ensure all teams utilize the tools in the best possible manner, which I would say is the primary objective of the guild"*[Developer]

The information security officer at Signicat was the guild lead, and the guild was composed of around 15 developers. Before each meeting, the security officer prepared the agenda, including topics such as vulnerability, risk, and security. To stay up-to-date within the security field, the guild could also discuss recently published news of interest, and all guild members could propose topics for inclusion.

TechDoc Guild

TechDoc (abbreviation for technical documentation) comprised four technical communicators, led by the Head of Product Experience at Signicat. A substantial portion of the workload was devoted to writing developer documentation for Signicat's developer sites. However, they also write and maintain product literature, create style guides and templates for documentation, manage product language translations, and create and maintain company naming conventions. To facilitate the handling of documentation as an iterative task, each communicator was assigned several agile teams, and they could attend all agile meetings of their respective teams. The communicators collaborated closely with the developers by using the same tools and processes for producing documentation that the developers use to produce code (the docs-as-code approach). The POs prioritized documentation tasks as it did for all other tasks, and most documentation tasks were submitted through each team's sprint or Kanban board. The guild maintained a dedicated Kanban board for documentation tasks outside agile teams. To ensure transparency, the guild operated a comprehensive confluence page which included information about established standards and processes, ongoing projects, meeting notes, and an informative and open-access landing page with key information on how to reach out to the guild.

Tech Excellence Guild

Signicat's tech excellence guild helped agile teams adopt best practices on agile principles, as well as facilitated the reporting of relevant KPI 's for each team within the monthly common retrospective meeting (presented in Section 4.3). Initially, the guild was accelerated during the early phases of the organizational transformation (see Section 4.1) as a tech excellence program between October 2020-March 2021. However, it has continued as a regular business since then. The guild was composed of SMs/agile coaches and some POs, making a total of 20 members, roughly. The members reported their respective team's KPI 's from the latest iterations at a monthly common retrospective meeting. Most teams ran two sprints (Scrum teams) or two iterations (Kanban teams) a month, but Nordic eID, which ran three-week sprints, had their SM adjusting the KPI 's within her spreadsheet, recalculating the team's 3-weekly efforts to fit the four-week cadence of the guild. The tech excellence guild did not have a dedicated lead during my observation period, so technology site leaders chaired the common retrospectives in a round-robin fashion. The guild managed a Confluence page storing all the common retrospective meetings and the aggregated KPI 's, as well as a knowledge-sharing base divided into "agile tips and tricks" and "agile help wanted ." The first contained tips and tricks on topics to be presented (or previously presented) in a lightning talk, whereas the latter contained topics guild members sought advice on. All guild members could propose topics. Table 4.3 presents the structure of the "tips and tricks" table and two examples of topics presented by guild members. The "help wanted" table was of a similar format. However, more of the time could have been spent on these activities: *"for the SMs, it is great to have some exchanges of ideas because I believe you may learn more from your own company than from random blogs on the internet"*[SM].

UX Guild

The user experience (UX) guild was led by the head of product experience at Signicat. It consisted of a UX lead and five UX designers who were assigned different agile teams as distributed UX resources (see

Topic	Team methodology	Description	Contact	Presented on
Make your epics smaller	Scrum	Smaller epics help the team focus	@[person1] @[person2]	dd.mm.yyyy
Retro.io for retrospectives	Scrum	Very easy to use, export and look back into past action items	@[person1] @[person2]	dd.mm.yyyy

Table 4.3: Tips & tricks topics presented in the Tech Excellence guild

Figure 4.6). In addition to sharing best practices within the UX domain, the guild managed checklists for developers, QA employees, and designers to ensure they covered all of the basic accessibility requirements relating to their work roles. Confluence was used by the UX team to store their work and larger UX projects. During the observation period, one of the main ongoing projects of the UX guild was creating a comprehensive design system that would bring together a common library that is easy to use by the agile teams and the marketing department. That said, I did not observe their activities closely. The information conveyed regarding the guild and the responsibilities of the UX designers is mostly based on information obtained from interviews and the guild's Confluence pages.

4.5 Inter-team Coordination Tools and Artifacts

Tools and artifacts are the last types of mechanism in the inter-team coordination taxonomy of Berntzen et al.. In line with the taxonomy, I have considered these to be objects that manage dependencies between development activities across teams. It can be material entities (tangible) and digital entities (intangible). All identified tools and artifacts supported coordination across teams by managing resource dependencies, either entity and/or technical dependencies. Also, they all managed knowledge dependencies, either due to the tools' nature of being collaborative facilitators (Slack and e-mail) or due to the fact that they contain knowledge and information per se. In total, I identified two main types of tools: communication tools and documentation tools. The latter includes artifacts: OKRs, burndown charts, roadmaps, knowledge bases, task boards, and product backlogs. In the following, I describe the usage of Slack for managing inter-team dependencies in more detail. This tool was used extensively by almost all agile team members to coordinate within and across teams and sites.

Communication tool: Slack

Slack seemed to be the most frequently used platform for communication across teams, tribes, and sites among several tools used for digital communication at Signicat (e.g., Outlook and Zoom). Slack enabled individuals and teams to communicate in an ad-hoc and rapid manner, contributing to knowledge dependency management. *"We have always used Slack, though at first mostly within the team only, but now you have a lot of channels across teams"* [Developer]. Apart from written communication in open spaces, Slack was also used for file sharing and private instant messaging, as well as for creating automated bots known as Slackbots that assisted in various tasks, such as alerting production errors. A wide variety of chat spaces were available, both open and private, known as channels, most of which were open team channels and specific channels dedicated to inter-team coordination. *All teams have their open channel that external to the team can use, and there is a lot of activity in the channels"* [Developer]. Such open channels helped transparency, allowing employees to get insights into other teams and products. Also, for instance, the POs had their channel for coordination of work related to the technology transformation (presented in Section 4.1). The TOPS framework presents Slack as a technical tool, given that most communication is driven by product development. However, Slack also supported social needs by connecting people, particularly teams at different sites and if some team members were working remotely.

5 Discussion

My research question is "How to balance alignment and autonomy in large-scale agile software development?". In line with Dingsøyr et al. (2018a), I defined large-scale as "development efforts that involve a large number of actors, a large number of systems and interdependencies, which have more than two teams." Development efforts at such scale have raised a set of challenges, including lack of alignment among teams and issues regarding decision-making efficiency (Dingsøyr et al., 2019). Previous research has described how well-functioning community of practice (CoP) preserve alignment and team autonomy at scale (Paasivaara and Lassenius, 2014; Smite et al., 2019; Paasivaara and Lassenius, 2019), but the research on how to best achieve a balance between the need for alignment and the need for teams to be autonomous is scarce. With this study, I contribute to filling this gap.

Large-scale agile frameworks have been created to address the challenges associated with software development at scale, and implementations of such frameworks are increasingly prevalent in contemporary software development organizations. However, the industry lacks contextually relevant advice from researchers to scale agile with these frameworks (Dingsøyr et al., 2019). I therefore decided to conduct a case study at Signicat, a leading provider of digital identity solutions in Europe. Similar to Ericsson's "agile journey" (Paasivaara et al., 2018), Signicat has gone through an agile transformation, or "tech excellence program," as Signicat called it. Signicat decided to employ a tribe structure and to unify their several technology platforms (see Section 4.1) to prepare for further growth and to better align their current agile teams and software products. My investigation of Signicat has resulted in a thorough description of Signicat and their transformation, identification and description of 12 inter-team coordination mechanisms (see Table 4.2), and a model for ascertaining the level of team autonomy on four dimensions.

In "Comparing Methods for Large-Scale Agile Software Development: A Systematic Literature Review," Edison et al. (2021) identified a total of 31 challenges across nine main categories associated with the use of large-scale agile frameworks. In the following, I discuss how to mitigate one team-related challenge and two inter-team coordination challenges, namely 1) lack of team autonomy, 2) synchronizing across dynamic and fast-moving teams, and 3) maintaining transparency across a high number of fast-moving, adaptive teams and projects. Addressing these challenges help me answer my research question.

5.1 Team autonomy

Providing teams with decision-making authority increases the speed and accuracy of problem-solving (Moe et al., 2009) and may offer several other advantages compared to traditionally managed teams. However, full team autonomy does not seem to be a realistic approach to software development at scale because teams in such environments do not deliver their work outcomes alone (Moe et al., 2021). Neither does full autocracy of management seem to be a valid approach to authority distribution as management-taken decisions within such contexts can *"easily fail to recognize the needs and the complexity of the work performed by the individual teams"* (Moe et al., 2021).

To understand the distribution of authority at Signicat, I collected data on team autonomy through a questionnaire distributed to all main technology sites of Signicat. In line with Moe et al. (2021), I have used the expression "level of autonomy" to refer to the amount of authority held by a team. I collected data on architecture, process, requirements, and tasks because I considered these to be important areas for software development teams. This is in line with how Moe et al. interpreted the functions of Hackman'

1986 authority matrix when they investigated the allocation of authority in large-scale agile (see Section 2.2.2). I could have investigated the autonomy level in several more areas. However, I found these four to be a reasonable pick given the limited time scope of this master's thesis (see Section 1.4). I also collected qualitative data on team autonomy. In the following, I discuss my findings in the four areas.

Architecture

Out of the 30 questionnaire respondents, only three disagreed or strongly disagreed with the statement that *"It is the team itself that makes the local architectural decisions"*, and none of the respondents chose autonomy on local architectural decisions as one of the three most important areas to improve at Signicat. It does not necessarily mean that architecture autonomy is not an area that should be improved at Signicat, but it is clearly one of the areas that need the least focus for further improvement. This goes well with the interview data, where a significant amount of the developers elaborated on how most of the architectural and technical choices were left to the team, and how wide-reaching decisions were left to management or the architecture guild. Several interviewees emphasized this as a reasonable balance between local decision authority and needed overarching bounds.

In the result section, I described how decentralizing architectural decisions, with the tech lead of each team as the main responsible for technical decisions made within each team, was beneficial for sharing architectural knowledge among teams. The collaborative team-based approach to architectural decision-making seemed to increase individuals' ability to see the big picture of the systems developed. Local technical consistency was preserved by distributing some of the responsibility to the tech lead and the developers. However, all developers emphasized the need for architects to be responsible for the *overall architecture* and for providing *guidelines*, which I further discuss in the following.

Overall architecture. At Signicat, architecture decision-making was decentralized to the operational level, with several tribe architects working together within an architect guild. The need for architects is supported by Dingsøyr et al. (2018a). Having tribe architects from each tribe operating together helped prevent the risk of changing feature prioritizations at each team or tribe causing major challenges in design, coding, and testing. It seemed to be beneficial in aligning architectural decisions across the company. This way of decentralization is supported by Salameh and Bass (2020), which found that instead of having a centralized group of architects responsible for the software architecture, the responsibility could be distributed among a set of architecture owners. The tribe architects at Signicat did not necessarily need to execute decision-making authority to ensure each team stayed in line with Sigicat's overarching architecture. Instead, the guild could facilitate overarching alignment with guidelines and recommendations. However, some decisions needed to be established as standards, whereas others could be left completely to the team themselves, thus prevailing team autonomy. Signicat's decision flow diagram for assessing architecture issues (see Figure 4.7) helped the architecture group to make sure the right decisions were made to ensure a balance between establishing standards and preserving team autonomy.

Guidelines. Architecture guidelines are a success factor for avoiding a divergent architecture at scale (Dingsøyr et al., 2018a,b). At Signicat, guidelines seemed to be beneficial for the development teams as an abstraction level of the software and for providing input on which programming languages are appropriate for which contexts, to be specific on one example. The question to be raised is how definite guidelines shall be. Should it be a list of recommendations on best practices, or should it be strict requirements? For instance, deciding which programming languages the teams are allowed to use is more than just deciding per se. It influences which people Signicat can hire and how easy or difficult it will be to transfer resources between teams. Allowing teams to decide on programming languages for specific applications in certain cases preserved autonomy on the team level, which seemed to raise the hunger for learning new technologies and programming languages.

On the other hand, to ensure a unified architecture, larger applications or components followed architectural principles and standards and were written in certain programming languages. Generally, one can understand that the teams are skeptical about must-use programming languages and would rather be able to decide for themselves. However, as one of the tribe architects emphasized, within a large and complex organization, there is a need for more guidelines and sometimes standards than the case for small businesses. It seems that Signicat had found the middle between extremes, which was highly appreciated among the interviewed developers at Signicat.

Process

The agile teams at Signicat ran either Scrum or Kanban. This is in line with Bjornson et al.'s (2018) suggestion to build on lightweight frameworks. All teams at Signicat had gone through the aforementioned "tech-excellence program," aiming to become more mature in the agile way of working, learning about best practices, and improving processes. However, the agile teams were allowed to adjust ceremonies, and add or remove arenas, which Hackman (1986) refers to as managing own work processes.

Both Team Signature and Team Nordic eID were adjusting their process during the observation period (e.g., shortening certain ceremonies or changing the frequency of a ceremony). Out of the 31 questionnaire respondents, 30 either strongly agreed (10) or agreed (20) that autonomy on processes was in place. Allowing teams to self-organize and decide how to implement agile practices is a reported success factor of large-scale agile transformations (Dikert et al., 2016; Edison et al., 2021). Albeit, Edison et al. suggests standardizing the way of working among teams as this enables people to be moved around easily and transferred between projects within an organization if needed (e.g., due to specialized competencies, employees leaving a company). Signicat's two methods at the team level (Scrum and Kanban), guidelines for team composition and roles (see Table 4.1), and the common cadence in the tech excellence guild ensured a satisfactory standardization across teams. So in parallel with allowing teams to decide their own agile practices, Signicat managed to standardize processes across teams: the tech excellence guild accelerated similar ways of working, and the tribe structure facilitated coordination and collaboration across teams working on inter-connected areas; but each team customized the daily way of working to fit their specific context, product, team size, and site distribution.

Requirements

Out of the four autonomy dimensions, requirements were the area the respondents differed most in their perception of the level of autonomy. The respondents clearly indicated that the teams do not have full decision authority on identifying, shaping, and prioritizing software requirements. That said, requirement specification and prioritization may not be an area in which developers should have a too large stake. A developer in team Signature emphasized that his job was to develop Signicat's signature product, and not *"to sit and prioritize as that's not my job"*. Requirements engineering is its own discipline and is by Edison et al. (2021) mentioned along with inter-team coordination and team-related challenges as one of the challenging areas within large-scale agile software development, including coordination of rapidly changing requirements across teams and capturing value and prioritization. Those challenges were seen at Signicat as well. The country product managers (CPMs) complained about the lack of a similar prioritization refinement process across the company. The refinement meeting of Team Nordic eID's PO and Signicat's CPMs worked properly as an arena for discussing and elaborating on prioritization across geographical markets. Team Signature did not have such an arena and emphasized the need to establish a similar arena to better communicate and refine requirements and prioritizations.

The point of discussing these aspects is to illustrate that it is not necessarily a lack of autonomy on requirements, as properly deciding on requirements demands a strategical view. The developers of both Team Nordic eID and team Signature participated in team refinement meetings and were invited to discussions when needed. The developers got insights on all issues and prioritizations, which led to more knowledge shared among participants. More *autonomy* on software requirements may not be needed, but improvements on the requirements engineering process *per se* are more likely to be beneficial.

Tasks

Team autonomy on task execution, referred to as "executing the task" by Hackman (1986), prevailed at Signicat. Most respondents either agreed or strongly agreed, and only four had it on their list of areas to improve. This impression was supported during the observations: both teams sat their sprint or iteration plans (with some inputs on prioritization from team externals as previously discussed), and the developers chose which tasks to work on. Additionally, the developers mostly made their own decisions on executing and solving tasks. That said, there were areas and features certain developers typically tended towards picking. "You touch it, you own it" was mentioned by some interviewees, indicating an implicit allocation of work. An advantage of such allocation is that developers are likely to build expertise within a specific domain of a team's responsibility, increasing coding efficiency and quality. A downside is the possible "bus factor effect," as one may risk depending on certain developers on certain segments of work, which the team may suffer from if a developer gets ill or leaves the team/company. However, this discussion concerns developer autonomy (individual autonomy) rather than team autonomy. The intention is to illustrate how developer autonomy may be fruitful and risky to the team and the organization. At the team level, autonomy on tasks was present.

5.2 Synchronization

Synchronizing the efforts of several teams when scaling agile has proven challenging (Paasivaara et al., 2018). Increased product complexity and intricate dependencies demand dependency awareness and aligned planning activities among teams (Bick et al., 2018). At Signicat, formal and informal arenas were in place to manage inter-team dependencies. Most meetings provided personnel access to information, particularly the unscheduled ones, facilitating quick coordination and decision-making. This is similar to the findings of Bjornson et al. (2018), where formal areas often finalized decisions, whereas informal and direct coordination between members of different teams made sure information was received and interpreted correctly across teams. Unscheduled meetings have also been demonstrated as a proper arena to resolve emerging coordination in several other studies (Dingsøyr et al., 2018a; Moe et al., 2018). Co-location and open landscape enabled the unscheduled meetings at Signicat. This is in line with Dingsøyr et al. (2018a) and Bjornson et al. (2018) which identified physical proximity to lowering the threshold of initiating ad-hoc contact. However, physical proximity was not a limiting factor for informal ad-hoc chats. At Signicat, Slack made up an extensive part of the daily communication within and across teams, tribes, and sites. Supporting the findings at Entur (Berntzen et al., 2022), instant messaging within dedicated channels successfully supported knowledge sharing and contributed to alignment. The usage of Slack as a well-functioning coordination tool is supported by several other studies as well (Dingsøyr et al., 2018b,b; Stray and Moe, 2020).

In addition to unscheduled meetings, the teams at Signicat used a series of scheduled inter-team coordination. The catwalks, common retrospectives, and the yearly kick-off meeting contributed to managing dependencies by enabling knowledge sharing across teams. These first two arenas compare to the demos and retrospectives in the Perform Programme Dingsøyr et al. (2018a). Additionally, both team Nordic eID and team Signature had refinement meetings with either CPMs or tribe personnel to manage process dependencies, and all agile teams at Signicat had team members participating in guild meetings, which the literature more commonly refers to as CoP meetings (Smite et al., 2019). In line with research conducted by Paasivaara and colleagues (2014), Smite and colleagues (2019), and Berntzen and colleagues

(2022), CoP (guilds) were used to support inter-team coordination across a wide range of areas. Each guild served a clear purpose, and though the mandate and scope varied among the guilds, all guilds were left with decision authority on the matters they handled. This is highlighted as success criteria for CoP in previous studies on guilds as well (Smite et al., 2019; Paasivaara and Lassenius, 2019). Yet I merged the guilds at Signicat to one role in Table 4.2, it is not adequate to elaborate on them as one type of role since they served quite different purposes. Whereas the security guild was composed of agile team members and aligned best practices, the architecture guild, for instance, was composed of tribe architects whose main purpose was to align architecture and technically synchronize teams, tribes, and sites. These tribe architects compare to the program architects at Entur (Berntzen et al., 2022). Most interviewees highlighted Signicat's tribe architects as important personnel for managing technical dependencies. The tribe architects ensured technical consistency within and across tribes and coordinated teams' efforts through tech leads.

Despite all the different meetings, arenas, and tools and artifacts for inter-team coordination, the questionnaire results indicated great possibilities for improvement on synchronization. This was further supported by the observations and interviews, where it seemed to be a lack of dedicated arenas for coordinating daily and weekly efforts between teams, in particular resource dependencies, but also dependencies on task allocation and activities at Signicat: the guilds appeared to be well-functioning arenas primarily for aligning knowledge, tools and procedures; the catwalks were an arena for showcasing work, not an arena for teams to synchronise work; the common retrospectives seemed to be too focused on key performance indicator (KPI) trendlines to be an effective arena for synchronising teams' development efforts; the kick-off meeting was a company wide conference bringing clarity to all employees on Signicat's mission and further direction, not an arena for synchronising development efforts, albeit establishing a common vision and clarity of goals and direction should not be underestimated as a factor when striving to align and synchronize teams (Paasivaara et al., 2018); and neither did the refinement meetings with CPMs and the PO (Team Nordic eID), or with tribe personnel and the PO (Team Signature) contribute much to coordinate technical resource dependencies among teams.

Some of the main large-scale agile frameworks introduce dedicated events for frequent coordination across teams. With Large-scale Scrum (LeSS), it is common to implement so-called multi-team meetings for product backlog refinement or sprint planning, and Scrum-at-Scale (S@S) offers a scaled daily scrum for teams to more easily be responsive to emerging impediments. Implementing such arenas is supported by the literature as a success factor. Previous research has identified several inter-team coordination mechanisms focusing on coordinating task progression between teams. For example, and contrary to Signicat, one of the largest IT projects undertaken in Norway, the Perform Progamme at the Norwegian Public Service Pension Fund (the "Pension Fund"), implemented many arenas to coordinate work between the teams (Dingsøyr et al., 2012). For instance, Scrum of Scrums (SoS) within subprojects, so-called Metascrums at project level, and a technical corner for architectural briefings for the teams were successfully implemented. The SoS and Metascrums may compare to both the multi-team meetings in the LeSS framework and the scaled version of the daily scrum in the S@S framework. Albeit, Perform was composed of a combination of in-house personnel and external consultants from five companies. Such a context is likely to demand other coordination needs than product development settings with stable domain knowledge (Dingsøyr et al., 2019), just as the case is for Signicat (digital identity solutions). However, with Signicat's several company acquisitions over the last few years, and the ongoing organizational and technological transformations, it may not be accurate to currently consider Signicat a stable large-scale software development case with established and mature teams. Rather, Signicat may be an extreme case requiring several arenas to foster communication and coordination across sites.

SoS may be a proper arena to implement at Signicat to better coordinate the efforts across teams, tribes, and sites. However, wide-reaching SoS has shown to work poorly since it often involves too many participants and disjoint interests and concerns (Paasivaara et al., 2012). Contrary, more narrowed inter-team meetings with participants having joint goals and interests are seen to more likely be perceived as successful. This is also supported by Dingsøyr et al. (2018a) which identified SoS on the subproject level as successful arenas. At Signicat, this may compare to SoS within tribes or SoS across tribes in certain development projects spanning two or more tribes. This would allow product owners, scrum master/agile coaches and/or tribe leads to meet and discuss projects to ensure all know what is going on (tribe leads at Signicat may be compared, for instance, to the product manager role at Entur (Berntzen et al., 2022)).

5.3 Transparency

Transparency "improves awareness across an organization regarding the ongoing development, as well as incentives teams to deliver high-quality software and to promote collaboration, communication and knowledge sharing" (Edison et al., 2021). At Signicat, the rapid growth over the past few years with several company acquisitions around Europe seems to have diminished transparency, and the corona pandemic with a heavily increased amount of remote working may have been a worsening factor. In their research, Dingsøyr and colleagues (2018), as well as Bjornson et al. (2018), find co-location and open work area to facilitate the flow of communication and information. Co-location allowed team members to get insights into what was going on in other teams, and several interviewees emphasized returning to the office as important to foster transparency at Signicat. However, post-corona, and as a continental organization, Signicat cannot rely upon co-location and mechanisms with physical characteristics to achieve and retain transparency. Instant messaging with Slack was used extensively to exchange information across teams, thus increasing transparency. Among digital communication tools, Slack has recently been identified as useful for instant messaging and information sharing (Stray and Moe, 2020). Stray and Moe found Slack to increase transparency by supporting constant information sharing.

Also, documentation tools and artifacts worked as transparency enablers at Signicat, in contrast to Ericsson's lack of shared backlogs, which led to poor transparency (Paasivaara et al., 2018). At Signicat, knowledge-based information was shared on Confluence (documentation tool), and each agile team's product backlog and sprint backlog/Kanban board were accessible to all employees. The latter may be compared to the physical boards in the Perform Programme (Dingsøyr et al., 2018a) and at Entur (Berntzen et al., 2022). Albeit, the physical boards at Perform and Entur seemed to be more effective enablers for discussions and information sharing.

The tribe leads and tribe architects' overview of technical and business process dependencies made them important inter-team roles, and developers' access to them was important for obtaining information. The allocation of teams into product tribes facilitated fostering of relation networks between teams operating within the same product area, thus increasing transparency. Also, the guilds seemed to function properly as transparency enablers. As the CoP at Ericsson (Paasivaara and Lassenius, 2014), the guilds at Signicat had open-access wiki pages storing meeting notes and other material. Additionally, the guilds let personnel get insights into other teams and their ongoing projects, and the prominent social characteristic of guilds let employees get to better know each other. Meeting regularly is supported in previous studies as an enabler for employees to foster relationships (Bjornson et al., 2018; Dingsøyr et al., 2018a).

5.4 Implications for Practice and Theory

My study of Signicat generates several practical implications. Regarding team autonomy, I have found that:

- Teams should define, change and improve their own internal agile framework, ceremonies and processes. However, light methodologies such as Scrum or Kanban should be adopted.

- Architectural decision-making authority should be decentralized from management to an architecture group. However, team autonomy on architecture decisions, with the tech lead of each team as the main responsible, should be strived for.

- Software requirements can hardly be fully identified, shaped, and prioritized by teams. What matters most is to continually include all team members in the backlog's evolution. Developers may have a role in some of the requirement engineering, but the identification and prioritization should be left to the POs, PMs, and team external roles.

- Teams should decide on task execution. Here, it is rather important to be aware of developer autonomy, which may introduce the "bus factor effect." To contradict this risk, stand-up meetings and other agile team ceremonies should be held to help the team stay updated on each other's work.

The tribe structure surrounded development teams and prevented teams from working in silos. Additionally, I have identified the following regarding synchronization practitioners should pay attention to:

- CoP (guilds at Signicat) function as efficient arenas for aligning teams on tools and best practices, and they should be provided decision-making authority on the matters they handle.

- Unstable large-scale agile development settings demand dedicated arenas for scrum masters/agile coaches, POs, and PMs to synchronize and stay up to date on related teams' work. Such arenas should be kept as narrow and focused as possible to avoid too many participants and disjoint interests.

- Unscheduled meetings and instant messaging with Slack work well to resolve inter-team dependencies. However, it should not replace formal arenas but rather be a supplement. Whereas ad-hoc chats often resolve dependencies, formal arenas may hinder them at an earlier stage.

As a final implication for practice, I have identified two facilitators and enablers for transparency at scale:

- CoP, bringing personnel across the company together to meet and discuss certain topics and build relationships. Additionally, it fosters information sharing and openness. However, increasing transparency is a gradual process, not something done overnight, and it therefore demands continuous attention from management and the firm.

- Co-location, whenever it is realistic, should be strived for. However, the modern work life (accelerated by the COVID-19 pandemic) demands digital collaboration tools. Slack is a proper substitute for ad-hoc discussions for instant sharing of information.

Besides the practical implications, my work also has some implications for theory. I have developed a team autonomy model that can be further examined and enhanced by researchers. The model lets researchers evaluate teams' level of autonomy on four dimensions in large-scale agile software development settings. Also, the model is suited to compare the level of autonomy among several teams, projects, and organizations or to compare autonomy levels across time and geographical locations within the same organization.

Additionally, I have demonstrated the explanatory ability of Berntzen et al.'s 2022 inter-team coordination taxonomy categories by using the taxonomy to identify inter-team coordination mechanisms at Signicat. I have also demonstrated its usability.

5.5 Evaluations and Limitations

In this master's thesis, I sought to understand how to balance alignment and team autonomy in large-scale agile software development settings. I conducted a mixed-methods exploratory case study at Signicat. In the following, I evaluate and discuss the limitations of my findings. However, an evaluation of the research design is not included as this is provided in the method chapter (Chapter 3).

The conceptual framework adapted by Scheerer et al. (2014) and the framework proposed by Ven et al. (1976) were considered to be used for my analysis. However, I decided to utilize the novel taxonomy and TOPS framework proposed by Berntzen et al. (2022). The taxonomy contributes to earlier taxonomies on inter-team coordination by extending the focus to the inter-team level and the knowledge domain of large-scale agile. Additionally, the taxonomy is developed in the current global "work from anywhere" situation where many inter-personal meeting arenas have been replaced with digital spaces, which applies to Signicat. Summa summarum, I considered the taxonomy and TOPS framework proposed by Berntzen et al. to be a reasonable choice for framing my analysis of Signicat's inter-team coordination mechanisms. A more comprehensive assessment of inter-team coordination at Signicat should have also obtained data from the management parts of the organization. This would have provided deeper insights on strategical decisions affecting underlying aspects of inter-team coordination and team autonomy.

There may as well be limitations related to theory. In particular, the body of literature on team autonomy in large-scale agile is scarce. The recently published study by Moe et al. (2021), which applied Hackman's 1986 classification of unit authority when analyzing how much authority teams at large-scale agile have, has been used extensively. To substantiate the analysis on autonomy, I have used studies on how CoP with decision-making authority help decentralize authority frequently (Paasivaara and Lassenius, 2014; Smite et al., 2019; Paasivaara and Lassenius, 2019). Also, findings from Entur (Berntzen et al., 2022) and the Performe Programme (Dingsøyr et al., 2018a), in particular, have been extensively used for comparisons. Nevertheless, the reliability of my analysis and discussion of autonomy at Signicat suffer from relatively few studies of related work available for comparison. The body of literature on inter-team coordination is larger and thus has provided me a greater amount of relevant comparisons to my findings on inter-team coordination at Signicat.

6 Conclusion and Future Work

In this master's thesis, I addressed the research question *"How can alignment and autonomy be balanced in large-scale agile software development?"*. To answer the question, I conducted a mixed-methods case study at Signicat, a leading provider of digital identity solutions in Europa. I analyzed data from 35 hours of observation, 13 interviews, a questionnaire with 31 respondents, and various documents. I used the inter-team coordination taxonomy and TOPS framework proposed by Berntzen et al. (2022) to identify inter-team coordination mechanisms. In total, I identified 12 mechanisms.

As noted by Dingsøyr et al. (2019), large-scale agile software development is seldom easy, and this master's thesis has investigated three challenges identified by Edison et al. (2021) in-depth, namely team autonomy, synchronization, and transparency. The literature provides little contextually advice on how to scale agile practices and implement large-scale frameworks. My description of the software development setting at Signicat shows one way to organize agile teams in large-scale development. I show how a tribe structure is adopted and how decision-making authority was decentralized with several community of practice (CoP), called guilds at Signicat. Tribes and guilds are both core elements of the Spotify model, but Signicat has found its way of implementing these structures.

With this study, I confirmed that large-scale agile software development efforts require both alignment and autonomy. Guilds enable alignment and transparency, as well as foster autonomy. Tribes tear down silos and may facilitate the creation of proper lines of communication and collaboration among product-related development teams. However, well-functioning guilds need to be provided decision-making authority on the matters they handle, and tribes do not foster proper arenas for communication between teams out of the book. Additional inter-team practices should be adopted to utilize the benefits of the tribe structure. This applies particularly to unstable large-scale software development settings. Specifically, this master's thesis suggests product or project-specific Scrum of Scrums (SoS).

From my findings, I make three contributions to large-scale agile software development. First, I identify three practices for mitigating team synchronization and transparency challenges at scale: guide teams within the same product or feature area into groups (tribes); decentralize decision-making by fostering communities (guilds) with decision-making authority; and facilitate small inter-team meetings for teams that have joint goals, for instance, SoS. Second, I propose a model for team autonomy on four dimensions: process, requirements, architecture, and tasks. Last, I provide an actionable approach for using the team autonomy model by introducing four statements that I map to each dimension in the model.

With these contributions, I hope to advance the knowledge on how to succeed in balancing the need for organizational control with the need and willingness of teams to be autonomous. I have proposed some clarity on how to mitigate poor synchronization and transparency and lack of team autonomy. However, I do not provide definite answers to my research question but rather several implications for practice and theory that should interest software practitioners and researchers. The team autonomy model is a flexible approach for measuring the level of team autonomy on relevant aspects to software engineering teams. Future studies may enhance the model by extending it with other dimensions, and I encourage future research to use the model for measuring team autonomy of software development teams at scale.

Further, the team autonomy model may support researchers and agile coaches and leaders in tracking changes in the level of team autonomy over time. Also, I encourage future studies to investigate how team autonomy and alignment can be balanced in more extreme contexts of distributed environments with employees primarily working remotely. After the outbreak of COVID-19, the work from anywherephilosophy has accelerated, which may demand other mechanisms for inter-team coordination to ensure transparency and team synchronization without depriving each team or each developer's autonomy.

Bibliography

7th State of Agile. Technical report, 2013. URL https://www.se.rit.edu/~swen-356/resources/7th-Annual-State-of-Agile-Development-Survey.pdf.

8th State of Agile. 2014. URL https://www.learningtree.com/Files/2013-state-of-agile-survey_LT.pdf.

9th State of Agile. Technical report, 2015. URL https://kipdf.com/state-of-agile-survey-9-th-annual_5afcc7c18ead0eb9108b4692.html.

10th State of Agile. Technical report, 2016. URL https://www.slideshare.net/Mateuszeromski/version-one-10thannualstateofagilereport.

11th State of Agile. Technical report, 2017. URL https://www.agile247.pl/wp-content/uploads/2017/04/versionone-11th-annual-state-of-agile-report.pdf.

12th State Of Agile. Technical report, 2018. URL http://www.eg.bucknell.edu/~cs479/common-files/resources/versionone-state-of-agile/versionone-12th-annual-state-of-agile-report.pdf.

13th State of Agile. Technical report, 2019. URL https://www.duxdiligens.com/wp-content/uploads/2019/09/13th-annual-state-of-agile-report_7_May_2019.pdf.

14th State of Agile. Technical report, 2020. URL https://www.qagile.pl/wp-content/uploads/2020/06/14th-annual-state-of-agile-report.pdf.

15th State of Agile. Technical report, 2021. URL https://digital.ai/resource-center/analyst-reports/state-of-agile-report.

Agile Alliance. Agile 101, 2021. URL https://www.agilealliance.org/agile101/.

Agile Alliance. Definition of Scrum of Scrums, 2022. URL https://www.agilealliance.org/glossary/scrum-of-scrums.

Deepika Badampudi, Samuel A. Fricker, and Moreno Ana M. Perspectives on Productivity and Delays in Large-Scale Agile Projects. In Barbara Baumeister Hubert
}and Weber, editor, *Agile Processes in Software Engineering and Extreme Programming*, pages 180–194, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-38314-4.

Corey Baham and Rudy Hirschheim. Issues, challenges, and a proposed theoretical core of agile software development research. *Information Systems Journal*, 2021. ISSN 13652575. doi: 10.1111/isj.12336.

Jordan Barlow, Justin Giboney, Mark Keith, Ryan Schuetzler, Paul Lowry, and Anthony Vance. Overview and Guidance on Agile Development in Large Organizations. *Communications of the Association for Information Systems*, 29:25–44, 7 2011. doi: 10.2139/ssrn.1909431.

K Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001. URL `https://agilemanifesto.org/iso/en/manifesto.html`.

Marthe Berntzen, Rashina Hoda, Nils B Moe, and Viktoria Stray. A Taxonomy of Inter-Team Coordination Mechanisms in Large-Scale Agile. *IEEE Transactions on Software Engineering*, PP(99):1, 2022. ISSN 0098-5589. doi: 10.1109/tse.2022.3160873.

Saskia Bick, Kai Spohrer, Rashina Hoda, Alexander Scheerer, and Armin Heinzl. Coordination Challenges in Large-Scale Software Development: A Case Study of Planning Misalignment in Hybrid Settings. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 44(10):932–950, 2018. ISSN 0098-5589. doi: 10.1109/tse.2017.2730870.

Finn Olav Bjornson, Julia Wijnmaalen, Christoph Johann Stettina, and Torgeir Dingsoyr. Inter-team Coordination in Large-Scale Agile Development: A Case Study of Three Enabling Mechanisms. In *AGILE PROCESSES IN SOFTWARE ENGINEERING AND EXTREME PROGRAMMING, XP 2018*, volume 314 of *Lecture Notes in Business Information Processing*, pages 216–231, 2018. ISBN 978-3-319-91602-6; 978-3-319-91601-9. doi: 10.1007/978-3-319-91602-6{_}15.

Kieran Conboy. Agility From First Principles: Reconstructing the Concept of Agility in Information Systems Development. *Information Systems Research*, 20, 7 2009. doi: 10.1287/isre.1090.0236.

Kim Dikert, Maria Paasivaara, and Casper Lassenius. Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, 119:87–108, 9 2016. ISSN 01641212. doi: 10.1016/j.jss.2016.06.013.

Torgeir Dingsøyr, Sridhar Nerur, Venu Gopal Balijepally, and Nils Brede Moe. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85 (6):1213–1221, 2012. ISSN 0164-1212. doi: 10.1016/j.jss.2012.02.033.

Torgeir Dingsøyr, Tor Erlend Fægri, and Juha Itkonen. What Is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development. In *Product-Focused Software Process Improvement*, pages 273–276, Cham, 2014. Springer International Publishing. ISBN 978-3-319-13835-0.

Torgeir Dingsøyr, Nils Brede Moe, Tor Erlend Faegri, and Eva Amdahl Seim. Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. *EMPIRICAL SOFTWARE ENGINEERING*, 23(1):490–520, 2018a. ISSN 1382-3256. doi: 10.1007/s10664-017-9524-2.

Torgeir Dingsøyr, Nils Brede Moe, and Eva Amdahl Seim. Coordinating Knowledge Work in Multiteam Programs. *Project Management Journal*, 49(6):64–77, 2018b. ISSN 8756-9728. doi: 10.1177/8756972818798980.

Torgeir Dingsøyr, Davide Falessi, and Ken Power. Agile Development at Scale: The Next Frontier. *IEEE Software*, 36(2):30–38, 2019. ISSN 0740-7459. doi: 10.1109/ms.2018.2884884.

Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9-10):833–859, 8 2008. ISSN 0950-5849. doi: 10.1016/J.INFSOF.2008.01.006.

Jutta Eckstein. *Agile software development in the large*. Dorset House, 2004. ISBN 978-3-947991-23-5.

Henry Edison, Xiaofeng Wang, and Kieran Conboy. Comparing Methods for Large-Scale Agile Software Development: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, PP(99): 1, 2021. ISSN 0098-5589. doi: 10.1109/tse.2021.3069039.

Sallyann Freudenberg and Helen Sharp. The Top 10 Burning Research Questions from Practitioners. *IEEE Software*, 27(5):8–9, 2010. doi: 10.1109/MS.2010.129.

J R Hackman. The psychology of self-management in organizations. In M S Pallack and R O Perloff, editors, *Psychology and work: Productivity, change, and employment.* American Psychological Association, Washington, DC, 1986.

James A Highsmith. *Agile software development ecosystems.* Addison-Wesley, 2002.

Rashina Hoda, Norsaremah Salleh, and John Grundy. The Rise and Evolution of Agile Software Development. *IEEE Software*, PP:1, 7 2018. doi: 10.1109/MS.2018.290111318.

Emam Hossain, Muhammad Ali Babar, and Hye-young Paik. Using Scrum in Global Software Development: A Systematic Literature Review. In *2009 Fourth IEEE International Conference on Global Software Engineering*, pages 175–184, 2009. doi: 10.1109/ICGSE.2009.25.

Michal Hron and Nikolaus Obwegeser. *Scrum in practice: an overview of Scrum adaptations.* ISBN 9780998133119. URL `http://hdl.handle.net/10125/50568`.

Jonas A Ingvaldsen and Monica Rolfsen. Autonomous work groups and the challenge of inter-group coordination. *Human Relations*, 65(7):861–881, 2012. doi: 10.1177/0018726712448203. URL `https://doi.org/10.1177/0018726712448203`.

Henrik Kniberg and Anders Ivarsson. Scaling Agile @ Spotify with Tribes, Squads, Chapters & Guilds. Technical report, 2012.

Claus W Langfred. The Downside of Self-Management: A Longitudinal Study of the Effects of Conflict on Trust, Autonomy, and Task Interdependence in Self-Managing Teams. Technical Report 4, 2007. URL `https://about.jstor.org/terms`.

Michael Y Lee and Amy C Edmondson. Self-managing organizations: Exploring the limits of less-hierarchical organizing. *Research in Organizational Behavior*, 37:35–58, 2017. ISSN 0191-3085. doi: 10.1016/j.riob.2017.10.002.

Thomas Malone and Kevin Crowston. The Interdisciplinary Study of Coordination. *Massachusetts Institute of Technology (MIT), Sloan School of Management, Working papers*, 26, 7 1993. doi: 10.1145/174666.174668.

Philipp Mayring. Qualitative Content Analysis. Technical report, 2000. URL `http://www.zuma-mannheim.de/research/en/methods/textanalysis/`.

Nils Moe, Torgeir Dingsøyr, and Tore Dybå. Understanding Self-Organizing Teams in Agile Software Development. In *Proceedings of the Australian Software Engineering Conference, ASWEC*, pages 76–85, 7 2008. ISBN 978-0-7695-3100-7. doi: 10.1109/ASWEC.2008.4483195.

Nils Brede Moe, Torgeir Dingsøyr, and Tore Dybå. Overcoming Barriers to Self-Management in Software Teams. *IEEE Software*, 26(6):20–26, 2009. ISSN 07407459. doi: 10.1109/MS.2009.182.

Nils Brede Moe, Torgeir Dingsøyr, and Tore Dybå. A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology*, 52(5):480–491, 2010. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2009.11.004. URL `https://www.sciencedirect.com/science/article/pii/S0950584909002043`.

Nils Brede Moe, Helena Holmstrom Olsson, and Torgeir Dingsøyr. Trends in Large-Scale Agile Development: A Summary of the 4th Workshop at XP2016. In *PROCEEDINGS OF THE XP2016 SCIENTIFIC WORKSHOPS*, Proceedings of the Scientific Workshop Proceedings of XP2016, pages 1–4, 2016. ISBN 978-1-4503-4134-9. doi: 10.1145/2962695.2962696.

Nils Brede Moe, Torgeir Dingsøyr, and Knut Rolland. To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software }development. *IJISPM-INTERNATIONAL JOURNAL OF INFORMATION SYSTEMS AND PROJECT MANAGEMENT*, 6(3):45–59, 2018. ISSN 2182-7796. doi: 10.12821/ijispm060303.

Nils Brede Moe, Darja Šmite, Maria Paasivaara, and Casper Lassenius. Finding the sweet spot for organizational control and team autonomy in large-scale agile software development. *Empirical Software Engineering*, 26(5):101, 2021. ISSN 1382-3256. doi: 10.1007/s10664-021-09967-3.

Briony J Oates. Researching Information Systems and Computing. Technical report, 2006.

Maria Paasivaara and Casper Lassenius. Communities of practice in a large distributed agile software development organization – Case Ericsson. *Information and Software Technology*, 56(12):1556–1577, 2014. ISSN 0950-5849. doi: 10.1016/j.infsof.2014.06.008.

Maria Paasivaara and Casper Lassenius. Empower Your Agile Organization: Community-Based Decision Making in Large-Scale Agile Development at Ericsson. *IEEE Software*, 36:64–69, 7 2019. doi: 10.1109/MS.2018.2886827.

Maria Paasivaara, Casper Lassenius, and Ville T Heikkilä. Inter-team Coordination in Large-Scale Globally Distributed Scrum: Do Scrum-of-Scrums Really Work? *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 235–238, 2012. doi: 10.1145/2372251.2372294.

Maria Paasivaara, Benjamin Behm, Casper Lassenius, and Minna Hallikainen. Large-scale agile transformation at Ericsson: a case study. *Empirical Software Engineering*, 23(5):2550–2596, 10 2018. ISSN 15737616. doi: 10.1007/s10664-017-9555-8.

Kai Petersen and Claes Wohlin. The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study. *Empirical Software Engineering*, 15(6):654–693, 12 2010. ISSN 13823256. doi: 10.1007/s10664-010-9136-6.

Knut Rolland, Brian Fitzgerald, Torgeir Dingsøyr, and Klaas-Jan Stol. Problematizing Agile in the Large: Alternative Assumptions for Large-Scale Agile Development Completed Research Paper, 7 2016.

Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 4 2009. ISSN 13823256. doi: 10.1007/s10664-008-9102-8.

Abdallah Salameh and Julian Bass. Influential Factors of Aligning Spotify Squads in Mission-Critical and Offshore Projects - A longitudinal embedded case study, 7 2018.

Abdallah Salameh and Julian Bass. Spotify Tailoring for Promoting Effectiveness in Cross-Functional Autonomous Squads. pages 20–28. 7 2019. ISBN 978-3-030-30125-5. doi: 10.1007/978-3-030-30126-2{_}3.

Abdallah Salameh and Julian M. Bass. Heterogeneous Tailoring Approach Using the Spotify Model. In *PervasiveHealth: Pervasive Computing Technologies for Healthcare*, pages 293–298. ICST, 4 2020. ISBN 9781450377317. doi: 10.1145/3383219.3383251.

Abdallah Salameh and Julian M Bass. An architecture governance approach for Agile development by tailoring the Spotify model. *AI & SOCIETY*, pages 1–20, 2021. ISSN 0951-5666. doi: 10.1007/s00146-021-01240-x.

Alexander Scheerer, Tobias Hildenbrand, and Thomas Kude. Coordination in Large-Scale Agile Software Development: A Multiteam Systems Perspective. In *2014 47TH HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES (HICSS)*, Proceedings of the Annual Hawaii International Conference on System Sciences, pages 4780–4788, 2014. ISBN 978-1-4799-2504-9. doi: 10.1109/hicss.2014.587.

Margrit Schreier. *Qualitative content analysis.* SAGE Publications Ltd, 2013.

Ken Schwaber and Jeff Sutherland. The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game. Technical report, 2020.

Mali Senapathi and Meghann Drury-Grogan. Systems Thinking Approach to Implementing Kanban: A Case Study. *Journal of Software: Evolution and Process*, 33, 7 2020. doi: 10.1002/smr.2322.

Dag I K Sjøberg, Anders Johnsen, and Jørgen Solberg. Quantifying the Effect of Using Kanban vs. Scrum: A Case Study 1. Technical report.

Darja Šmite, Claes Wohlin, Tony Gorschek, and Robert Feldt. Empirical evidence in global software engineering: A systematic review. *Empirical Software Engineering*, 15(1):91–118, 2 2010. ISSN 13823256. doi: 10.1007/s10664-009-9123-y.

Darja Smite, Nils Brede Moe, Georgiana Levinta, and Marcin Floryan. Spotify Guilds: How to Succeed With Knowledge Sharing in Large-Scale Agile Organizations. *IEEE Software*, 36(2):51–57, 2019. ISSN 0740-7459. doi: 10.1109/ms.2018.2886178.

By Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj. Challenges of migrating to agile methodologies. Technical Report 5, 2005.

Viktoria Stray and Nils Brede Moe. Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and Slack. *Journal of Systems and Software*, 170:110717, 2020. ISSN 0164-1212. doi: 10.1016/j.jss.2020.110717.

Viktoria Stray, Nils Brede Moe, and Rashina Hoda. Autonomous agile teams. *Proceedings of the 19th International Conference on Agile Software Development: Companion*, pages 1–5, 2018. doi: 10.1145/3234152.3234182.

Diane E Strode. A dependency taxonomy for agile software development projects. *Information Systems Frontiers*, 18(1):23–46, 2016. ISSN 1387-3326. doi: 10.1007/s10796-015-9574-1.

Diane E Strode, Beverley Hope, Sid Huff, Sebastian Link, Diane E ; Strode, Beverley ; Hope, Sid ; Huff, and Sid L Huff. *Coordination Effectiveness In An Agile Software Development Context.* 2011. ISBN 978-1-86435-644-1. URL `http://aisel.aisnet.org/pacis2011/183`.

Andrew H Van De Ven, Andre L Delbecq, and Richard Koenig. Determinants of Coordination Modes within Organizations. *American Sociological Review*, 41(2):322, 1976. ISSN 0003-1224. doi: 10.2307/2094477.

RK Yin. *Case Study Research: design and methods*. Sage, 5 edition, 2014.

Appendices

A Case study proposal

Following the first meeting with Signicat representatives fall 2021, I wrote a case study proposal (in Norwegian) including suggestions for data collection. See the proposal below.

Tanker om tematikk, vinkling og mulig spissing
Med storskala smidig utvikling og en struktur inspirert av Spotifymodellen med tribestruktur med profitloss-ansvar på tribenivå som bakteppe, så kunne en mulig interessant vinkling være å se på hvordan koordinering av mange team balanseres med selvstyre innad i hvert enkelt team. Med andre ord hvordan autonomi innad og på squadnivå lar seg forene med styringsbehov på tribenivå og eventuelt andre høyere nivåer.

Gitt en slik vinkling, vil en mulig spissing rettet mot eksempelvis enten arkitekturarbeid, prosess eller kravspec kunne være aktuelt og interessant, for å ta tre eksempler. Her er det vanskelig å nødvendigvis peke ut ett av forslagene allerede nå. Det er mye mulig at det vil være hensiktsmessig å avvente dette til ut på nyåret etter en evt. observasjonsstart, samt etter å ha rukket en viss opplesing på bakgrunnsstoff på både caset og på fagfeltet.

Grovskissert mulig plan for våren
Observasjoner og kortere intervjuer fremstår som interessante aktiviteter å kunne få til og gjennomføre. Et forslag er å kjøre to runder atskilt av en del ukers opphold for å 1) få tid til å transkribere og analysere funn, samt 2) la organisasjonen og strukturen få tid til å modne litt (hvor enn mye det er mulig på noen uker). Hvilke arenaer som egner seg (og som er aktuelle/tilgjengelige for meg) må vi eventuelt komme til enighet på, men tenker at det vil være hensiktsmessig å få til noe utover å være fysisk tilstede, selv om det også vil være nyttig på sine måter.

Kortere intervjuer kunne blitt gjennomført med medlemmer fra et par-tre utvalgte team, og da gjerne "ekstrem"-team som skiller seg litt fra hverandre (eksempelvis på metodikk, arbeidsområder i teckstacken etc.). Eventuelt kunne disse teamene også vært team jeg prioriterer i observasjoner. Hvorvidt man plukker ut disse teamene allerede til runde 1, eller om dette først skjer til runde to, kan vi se på (gitt at det blir aktuelt å rette fokuset inn mot et utvalg av to-tre team). Godt mulig at runde 1 bør/må benyttes til å se hvilke arenaer og hvilke team som kan være av ekstra interesse for runde 2.

Mhp. varighet av en "runde" med observasjoner/intervjuer, så kan en hel iterasjonslengde + eventuell en liten uke for å få med litt overlapp mellom to iterasjoner være en fornuftig tilnærming. Frekvensen på observasjoner i en periode kan være på alt fra kun én gang i uken til en litt høyere frekvens. Dette beror på hva som er praktisk mulig, hva som lar seg gjennomføre mhp. andre aktiviteter/planer hos meg, men også deres interesse av å ha tredjeparter tilstede.

Mer konkret, foreslår jeg følgende grovskisserte plan slik at dere lettere får en oversikt over hva omtrent jeg ser for meg:
Runde 1: Oppstart en gang mellom 17.jan-24.jan, slutt en gang mellom 14.feb-21.feb, to observasjonsdager i uka, og kortere intervju med utvalgte personer fra to team.
Runde 2: Oppstart: en gang mellom 7.mars-14.mars, slutt en gang mellom 28.mars-4.april, én observasjonsdag i uka, og kortere intervju med utvalgte personer fra to team.

Etter runde 2 (medio mai): Feedback-sesjon for legge fram “foreløpige” funn før planlagt innlevering av oppgave i starten av juni.

B Observation protocol

I made an observation protocol to standardize my meeting notes to simplify the process of analysing my meeting notes. See the protocol below.

Topic	Question
What	What meeting type is it?
Participants	Who are intended participants? Who leads the meeting?
Activities	What do they discuss?
Goals	What are they trying to accomplish with this meeting?
Feelings	Does the meeting work properly in order to accomplish the goal? How is the atmosphere?
Time	When does the meeting start? When does the meeting end? Delays?

C Interview

I Invitation

Interviewees were invited by e-mail (in Norwegian). Along with the invitation, an information letter and consent following the the standards of the Norwegian Centre for Research Data was attached. See the invitation template below.

Hei, [NAVN]

Jeg har de foregående ukene deltatt på mange av de agile seremoniene til dere i Team [TEAMNAVN], men det har du antagelig fått med deg alt. Som jeg annonserte på Slack forrige uke, så har jeg et ønske om å tillegg få til noen intervjuer med en del av dere. Jeg har lagt intervjuene til neste uke, og jeg kunne veldig gjerne tenkt meg å invitere deg selv om jeg antar at du har nok å henge fingrene i som utvikler.

Kort om intervjuet/praten

Intervjuet vil være en del av datagrunnlaget i masteroppgaven min i datateknologi dette semesteret. Temaet i oppgaven er hvordan behovet for koordinering på tvers av squads (og tribes) lar seg forene med, og evt. går utover, selvstyre på squadnivå. Intervjuet vil følgelig ta for seg temaer knyttet til dette. Det er ikke meningen at dette skal være de altfor omstendelige og formelle greiene, men snarere en “løs prat” rundt noen få tema slik som din rolle og seremoniene og arenaene dere i Team [TEAMNAVN] benytter i det daglige. Håper en slik prat er noe som virker ok å delta på, og at du både har tid og lyst.

Tidspunkt og sted

Har satt opp et forslag til tidspunkt neste uke. Det er bare å foreslå et annet tidspunkt den samme tirsdagen eller den påfølgende onsdagen om mitt forslag ikke passer. Hvis uke 10 er håpløst, så kan du foreslå et tidspunkt tirsdag/onsdag i uke 11 også. Jeg tar gjerne praten på Signicats lokaler i Trondheim (et av møterommene), men Zoom fungerer også fint om det heller er å foretrekke.

II Interview guide

The interviews were unstructured, but followed an interview guide approved by the Norwegian Centre for Research Data. The interview guide stated several categories, each holding a number of possible questions. All questions were formulated as general and unbiased as possible to avoid prompting the respondents towards providing already-determined answers. See the most used questions during the interviews below.

Part	**Question**
1. Introduction	Present myself and the project Thanks for participating Assure confidentiality Asks for permission to tape record
2. Warm up	What is you role and what projects do you work on? Tenure? How do you make plans? How are architectural choices agreed upon? Do you collaborate with people outside the team?
3. Synchronisation	Do you need to communicate with other teams? What kind of dependencies do you have to other teams? • Examples? • Consequences? How do you manage (and solve) dependencies? • Formal or informal? • Who manage them? • Examples? • Challenges? • Most important arena?
4. Transparency	Do you have practices for knowledge sharing? • How is it organized? • What type of knowledge do you share? • Decision-making authority? Do you know what other teams are working on?
Closing	Anything else you think we should have discussed? Thank you

D Questionnaire

The questionnaire contained four sections: an introduction; two sections with statements (each with a semi-introduction); and a section for the respondents to highlight improvement areas from eight pre-selected areas. Possible answers on the statements were from 1 - strongly disagree to 5 - strongly agree. See the questionnaire below.

Part	Question/Information
Introduction	You are asked to answer four question about team autonomy, and four questions about inter-team coordination (definitions of those two terms will be provided). You are asked to indicate the current state. The questionnaire is anonymous, and takes no more than 5-7 minutes to complete. The collected data will be part of the results of a master thesis in computer science at the Norwegian University of Science and Technology spring 2022. The thesis seeks to answer how software development organizations balance inter-team coordination and team autonomy
Team autonomy	Team autonomy refers to how much freedom a team has while it is working. Autonomy within the field of software engineering mean teams are allowed to decide on the priority, design and development of features, and to set their own scheduled and decide how their work should be done. The intentions is to see which decisions regarding day-to-day software engineering activities are made *by your team*, and which decision are made *outside your team.* • Agile process: It is the team itself that defines, changes and improves the internal agile framework, ceremonies and processes (how to work) • Software requirements: It is the team itself that identifies, shapes and prioritizes the software requirements (what shall be in the system) • Local architectural decisions: It is the team itself that makes the local architectural decisions (all kinds of technical choices) • Task execution: It is the team itself that decides how tasks should be executed and solved (how and by who)
Inter-team coordination	Inter-team coordination (ITC) is the coordination between two or more teams. Do not confuse it with intra-team coordination, which is coordination within a team. ITC involves synchronising the efforts of multiple teams and managing cross-team dependencies, and it can be done through scheduled and non-scheduled activities, formal as well as informal • Transparency: I am aware of other teams and their areas of responsibility • Dependency awareness: We are aware of dependencies to other teams • Team synchronisation: We get interrupted by other teams and their tasks • Time spent on ceremonies/meetings: I spend more time than necessary on agile ceremonies and other meetings
Improvement(s)	If there are any, will you point out the most important topics to improve at Signicat? (max. three out of the eight topics covered in this questionnaire). If necessary, go back to the question sections to get an overview of the covered topics • The most important • The second most important • The third most important