Master's thesis

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

Tommy Woldseth

# Teamwork Effectiveness in Large-Scale Agile Software Development: A Multi-Case Study

Master's thesis in Datateknologi
Supervisor: Torgeir Dingsøyr
June 2023

NTNU
Norwegian University of
Science and Technology

Tommy Woldseth

# Teamwork Effectiveness in Large-Scale Agile Software Development: A Multi-Case Study

Master's thesis in Datateknologi
Supervisor: Torgeir Dingsøyr
June 2023

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

NTNU
Norwegian University of
Science and Technology

# Abstract

Agile methods were originally designed for small, co-located software development teams. In recent times, however, agile methods have been increasingly applied to larger projects where multiple agile teams cooperate together. Such a large-scale context entails more teams, developers, and stakeholders, which can make coordination and collaboration more challenging. Therefore, this thesis investigates factors that contribute to effective teamwork in large-scale agile software development, both at the intra and inter-team level. To accomplish this, two large-scale agile cases were researched through the use of interviews. In total 14 interviews were conducted with developers, architects, team leads, and product leads. The statements from these interviews were analyzed qualitatively and used to attempt to answer three research questions regarding teamwork in large-scale agile. The first main finding consists of an overview of factors that foster and hinder effective teamwork in large-scale agile. The second main finding is a collection of strategies that can be employed to enable teamwork effectiveness. Lastly, the findings are compared to an existing model of teamwork effectiveness in agile teams, and an extension of the model is developed and proposed specifically for large-scale contexts.

# Sammendrag

Smidige metoder ble opprinnelig designet for små, samlokaliserte programvareutviklingsteam. I nyere tider har imidlertid smidige metoder i økende grad blitt brukt på større prosjekter hvor flere smidige team samarbeider. En slik stor-skala kontekst innebærer flere team, utviklere og stakeholdere, noe som kan gjøre koordinering og samarbeid mer utfordrende. Derfor undersøker denne oppgaven faktorer som bidrar til effektivt samarbeid i stor-skala smidig programvareutvikling, både på intra- og inter-team-nivå. For å oppnå dette ble to stor-skala smidige caser undersøkt gjennom bruk av intervjuer. Totalt ble det gjennomført 14 intervjuer med utviklere, arkitekter, teamledere og produktledere. Utsagnene fra disse intervjuene ble analysert kvalitativt og brukt til å forsøke å svare på tre forskningsspørsmål angående teamarbeid i stor-skala smidig. Det første hovedfunnet består av en oversikt over faktorer som fremmer og hindrer effektivt teamarbeid i stor-skala smidig. Det andre hovedfunnet er en samling av strategier som kan brukes for å muliggjøre effektivt samarbeid. Til slutt sammenlignes funnene med en eksisterende modell for effektivt samarbeid i smidig utvikling, og en utvidelse av modellen blir utviklet og foreslått spesielt for stor-skala sammenhenger.

# Preface

This thesis concludes my master's degree from the Department of Computer Science, as well as five fantastic years at NTNU. During these years I have met a lot of great people, made new friendships, learned a lot, and had a lot of fun.

Throughout my time at NTNU, I have taken a broad range of various courses. I first got introduced to agile development in the course TDT4140 - Software Engineering and have been interested in the field of software development processes ever since. Through this course, I was also introduced to Henrik Kniberg's down-to-earth book on Scrum and XP, which further sparked my interest and shaped my view on agile in general, and Scrum in particular (Kniberg 2015). I got familiar with the use of agile in large-scale contexts through a preparatory specialization project which was carried out the past fall. The project consisted of a literature review within large-scale agile software development, and the research made me want to continue investigating this topic for my final project.

Lastly, I want to thank some people who have helped me through this final project. First and foremost, I want to give a big thanks to my supervisor Torgeir Dingsøyr. Torgeir has guided me through the project from start to finish and has learned me a lot about software development processes, research, and academic writing. He has also provided invaluable feedback on my theses and helped me gain access to the two cases I researched in this project. Further, I want to thank the other members of my supervisor group, for useful tips, feedback, and discussions. I also want to thank the people at Signicat and NAV IT for taking the time to help me with my project and share their experiences. Lastly, I want to thank my family, friends, and girlfriend for always being supportive.

Tommy Eikrem Woldseth

*Trondheim, June 4, 2023*

# Table of Contents

# List of Figures

# List of Tables

# 1 Introduction

This initial section will introduce agile, and more specifically, large-scale agile software development. It will provide relevant background on key topics of the research as well as motivation for this study. Further, the research questions will be introduced, as well as a discussion of the contributions, intended audience, and limitations of this study. Lastly, the section will give an overview of the structure of this thesis.

## 1.1 Background and motivation

Agile software development has been a popular way of developing software since its inception in the late 90s (Dingsøyr, Falessi et al. 2019). Traditional, plan-based software development methodologies have in many cases been replaced by agile methods and an agile mindset. Agile methods were originally intended for small, co-located development teams (Dingsøyr and N. B. Moe 2013). Due to their success, however, agile methods have been increasingly employed in other contexts as well. In recent times, larger organizations have used agile methods in large projects where a multitude of development teams cooperate to develop software (Fuchs and Hess 2018). This has created a new sub-field within software development methodologies, namely large-scale agile software development. As this is a relatively new field within software development, it can benefit from more empirical research.

One example of large-scale agile development is the development program from the study of Dingsøyr, N. B. Moe and Seim (2018). In this study, a program called "Perform" was explored, which developed an office automation system for the Norwegian Public Service Pension Fund. The program employed an agile approach and was labeled as large-scale due to its size. At its peak, the program consisted of 12 teams working in parallel, involving 175 people in total. With a total budget of around EUR 140 million, and using about 800,000 person-hours, it was one of Norway's largest IT programs. A more detailed description of what constitutes "large-scale" in agile development will be presented in Section 2.1.4.

Developing software requires a substantial effort, and is hence developed by teams

where the work is shared amongst multiple team members. One aspect that is vital for all teams is being able to work together in an effective way and have a sufficient level of communication and coordination to keep productivity high. In large projects, the amount of work is too large for one team to be able to finish in an acceptable amount of time, hence multiple development teams who cooperate and coordinate the work amongst them are required. When there is a need to cooperate both at the team level and at the inter-team level, the importance of being able to work well together may grow even larger. Further, the use of agile methods may also increase the needs for effective teamwork. As the agile philosophy encourages an iterative approach to planning, less dependencies may be identified before they emerge. This may increase the need for a high level of coordination and teamwork to resolve unexpected challenges when they arise.

But what are the potential consequences of ineffective teamwork in large, agile projects? One instance of an IT project that did not go according to the plan is the P3 project from the Norwegian Labour and Welfare Administration (NAV). This project was supposed to modernize and automate the processing of various applications within the organization, regarding benefits such as sick pay. The project that started in 2018 was supposed to be finished in 2020 and the goal was to cut costs on manual processing by 61%[1]. However, as of now the development of the system is yet to be completed, and new forecasts estimate that the new system may not be ready until 2027[2]. The forecasts further estimate that the system that was supposed to be a cost-effective measure may produce a net cost of NOK 1.74 billion from 2021 to 2030, compared to costs before the project was started. Moreover, the new system has caused the wait time for processing of complaints to grow from 12 weeks to 52 weeks. The project has been evaluated by the consulting firm PwC, which found several significant weaknesses in the program. One of these weaknesses was struggles with teamwork in cross-functional teams[3]. While teamwork-related challenges were not the sole cause of the underperformance, they may have played a critical part in

[1]https://klassekampen.no/samling/navs-it-sprekk/2023-01-03/it-fiasko-gir-kjempeko1

[2]https://klassekampen.no/samling/navs-it-sprekk/2023-01-23/navs-prognose-it-smell-gir-174-mrd-i-minus

[3]https://klassekampen.no/utgave/2023-05-23/slaktes-for-it-trobbel

the outcomes of the project. Hence, research on the topic of teamwork effectiveness may help future large-scale projects to maintain a higher level of performance.

While practitioners are actively using agile methods in large-scale contexts, researchers have expressed the need for more empirical studies on the topic. At the XP 2010 conference, a group of about 300 practitioners was asked to vote on the top burning research questions within agile software development. At the top of the list was the topic of "Agile and large projects" (Freudenberg and Sharp 2010). Further, at the XP 2013 conference, Dingsøyr and N. B. Moe (2013) identified that there was significant interest in large-scale agile development. However, they also noted that few studies on the topic existed. Moreover, they suggested a research agenda for future research on large-scale agile software development, where the topic of "inter-team coordination" ranked first. This research thesis will provide a new study on the topic of large-scale agile and address a range of sub-topics, among them inter-team coordination.

## 1.2 Research questions and scope

This project will aim to explore how large-scale agile software development projects facilitate effective teamwork. It will attempt to find factors that promote effective teamwork, as well as factors that hinder effective teamwork in this context. The findings will then be compared with an existing model on teamwork, the *agile team effectiveness model* (ATEM), to investigate whether this model is also applicable in a large-scale context. In order to explore these topics, the following research questions have been defined:

- **RQ1:** What factors do participants perceive as impacting teamwork effectiveness in the large-scale agile context?
  - **RQ1.1:** What fosters effective teamwork?
  - **RQ1.2:** What hinders effective teamwork?
- **RQ2:** How do large-scale agile teams enable teamwork effectiveness?

- **RQ3:** How do the findings compare to the existing model on teamwork effectiveness, ATEM?

## 1.3 Contributions

This study will provide empirical research on the topic of large-scale agile software development, of which existing literature is sparse. Further, it provides a multi-case perspective by researching two separate cases, which is even more rare in the field. The research will include an insight into how large-scale agile software development projects facilitate effective teamwork. This includes methods, tools, routines, and processes that are used to improve how project members cooperate, both on the intra-team and inter-team levels. This insight can aid other agile projects in determining how to employ agile at scale. Additionally, this study will provide an overview of factors that promote effective teamwork, and factors that hinder effective teamwork. Such an overview can be used by practitioners to find new ways to improve teamwork in their projects. The overview can also be of interest to other researchers, who could conduct similar research in other organizations or contexts and compare the findings. Lastly, the findings of this research will be compared to an existing model on teamwork from the literature, the ATEM model (D. Strode, Dingsøyr et al. 2022). This can further confirm this model in a new context, or provide evidence that the model does not suit large-scale agile software development well. In conclusion, this study can be of interest both to practitioners with the intent of increasing productivity through improving teamwork, and to researchers who aim to further the research on teamwork in large-scale agile development.

## 1.4 Intended audience

This study is a master's thesis as part of a computer science degree for the Department of Computer Science at the Norwegian University of Science and Technology. As such, the thesis will be read and graded by a master project supervisor in addition to an internal and an external examiner. Further, the thesis can be of interest to practitioners and researchers within the Software Engineering and Information

Systems fields. It is expected that the reader has a background in computer science, either from the industry or academia, and is familiar with common software development processes and methodologies. For readers that lack relevant background knowledge, it is recommended to read Section 2, which will provide an extensive description of the relevant background theory, before continuing with the rest of the thesis.

## 1.5 Limitations

As this study is a master's thesis, the research period is limited to one semester. This prevents the research from investigating longitudinal effects that can be found over a longer time period. In addition, the research will be conducted by one researcher. This means that the data will only be retrieved and analyzed by one individual, which prevents investigator triangulation (Oates 2006). The fact that the research is confined within the limits of a master thesis also restricts the number of work hours that are available for the project, which is expected to be around 800 hours[4].

## 1.6 Structure of the thesis

This research thesis will be divided into six main sections. Below follows a description of the contents of each of these sections.

**Section 1 - Introduction**
Introduces the thesis and its key topics. This includes providing some background on the topic to be researched, as well as providing motivation for why this research is selected. Further, this initial section introduces the research questions that the thesis will attempt to answer, as well as discusses the contributions, intended audience, and limitations of the research.

[4]https://www.mastersportal.com/articles/388/all-you-need-to-know-about-the-european-credit-system-ects.html

**Section 2 - Background and Theory**

Provides an overview of relevant existing theory and literature related to this study. This includes theory on key topics such as agile software development and teamwork. Other relevant topics such as remote work and knowledge management will also be covered.

**Section 3 - Method**

Describes how the research of this project has been conducted. It will present the selected research strategy, as well as describe the research design phase, case selection, relevant data generation methods, and the analysis phase of the project. Additionally, it will assess the validity of the selected research strategy.

**Section 4 - Results**

Presents the findings coming from the data generation. Key findings will be thematically presented according to categories identified during the analysis of the data. Findings from the two cases will be presented together so that differences and similarities can be highlighted.

**Section 5 - Discussion**

Discusses the findings from Section 4 in light of the research questions from Section 1. Each research question will be presented and discussed before they will be answered in order.

**Section 6 - Conclusion**

Brings a conclusion to the research questions, based on the results and discussion. It will also bring light to the contributions this study has produced, as well as suggestions for topics that require further research.

# 2 Background and Theory

This section will present background, theories, and literature on topics that are relevant to the aim of this thesis. That includes a description of software development processes, theories on teamwork and coordination, as well as other relevant background material. It will also describe and define terms that will be used later in the thesis.

## 2.1 Software development processes and models

In a software development project, the main goal is to develop some type of software, either for internal use, for clients, or for customers. In order to do this, teams consisting of developers, designers, testers, and team leads, among other potential roles, cooperate to be able to reach the project goals. To ease collaboration and enable a systematic way of working, such teams may need to follow specific processes. Such processes impose structure on software engineering, with the aim of making development systematic, repeatable, and more success-oriented (Bourque et al. 2014). A software process is a set of activities that leads to the production of software (Ian Sommerville 2010). These activities can include elements such as practices, routines, or the use of specific tools at specific times. Software processes can be classified as either plan-based or agile, which will be described further later in this section. A range of various software processes exists, but they all include four fundamental activities, according to Ian Sommerville (2010). These are *software specification*, *software design and implementation*, *software validation*, and *software evolution*. As these are relatively broad terms they consist of various sub-activities, such as requirements engineering, architectural design, testing, and so on.

A software process model is a simplified representation of a software process (Ian Sommerville 2010). Such a model can describe the development life cycle of a software product. The life cycle often includes various stages of development, such as planning, design, implementation, and testing. Plan-based models tend to include thorough up-front planning before the implementation phase. Agile processes, on

the other hand, use an iterative approach, where planning and implementation are done in a cyclical rather than sequential fashion. Next, plan-based methods will be described further.

### 2.1.1 Plan-based methodologies

Plan-based software methods have been labeled "heavyweight" and "traditional", compared to newer methods (Dybå and Dingsøyr 2009; Girma et al. 2019). These methods tend to have a clear separation between phases in the software development life cycle, where one phase is completed before moving on to the next phase. Extensive planning of the development is done up-front, which entails well-documented requirements and tasks before the implementation phase. A drawback of this approach, however, is the inability to adapt and change course after the implementation has started, as most of the planning has already been completed. The cost of changes increases over time, as more and more assumptions and decisions about the final product are made. This way of developing software also implies a longer wait for an initial prototype, compared to agile methods.

#### Waterfall model

The waterfall model has been a widely used software development process since its inception in the 70s (Petersen et al. 2009). The name "waterfall" comes from the way that the phases cascade onto the next as seen in Figure 1. The five stages are the following, as defined by Ian Sommerville (2010):

1. **Requirements definition:** The services, constraints and goals of a system are established, and then defined in detail to serve as a system specification.

2. **System and software design:** The overall system architecture is developed. Abstractions of the software system and their relations are identified and described.

3. **Implementation and unit testing:** The software is developed as a set of programs or program units. Unit testing is used to verify that the units meet

their specification.

4. **Integration and system testing:** Individual parts of the program are integrated and tested as a complete system to ensure that the requirements are met. After testing, the system is delivered to the customer.

5. **Operation and maintenance:** Typically the longest phase of the life cycle. The system is put to practical use by its intended users. Maintenance involves fixing errors that have not already been identified and improving the implementation and the system's services as new requirements are discovered.

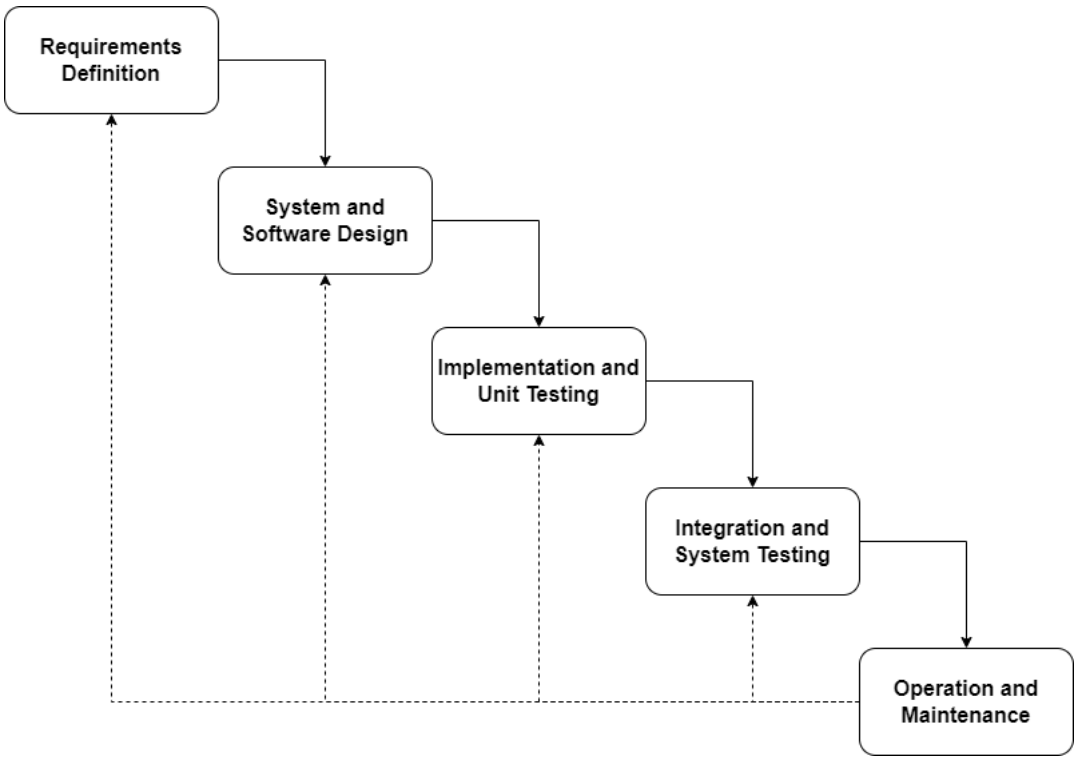



Figure 1: The waterfall model, illustration inspired by (Ian Sommerville 2010)

The longer planning phase at the start of the project allows for well-defined requirements, system architecture, and graphical user interface design. This can enable a faster pace of implementation, as engineers know what to build, how components should interact with each other, and how the software program should look. However, this linear approach to the phases of software development has its drawbacks. As most of the planning is done before the implementation begins, a project can spend multiple months planning before writing a single line of code. During the

time it takes to plan, implement, and test a solution, the project's requirements may change due to changes in the industry, the market, legislation, or in general customer needs (Fuchs and Hess 2018). When changes in the requirements occur, parts of the planning that has already been done may have gone to waste and the product that is built may not be what it is supposed to be. This can be costly to clients or stakeholders in terms of time and money, as the project may need a new iteration of planning, designing, implementation, integration, and testing.

In principle, each stage should be finished before moving on to the next one. Due to the inflexibility concerning requirement changes, the waterfall model is best suited when the requirements are clear and unlikely to change during development. However, from a management point of view, it may be easier to use the same process for an entire organization instead of using a separate model for software development (Ian Sommerville 2010). This can result in the use of the waterfall model in software development projects even when it is likely that requirements can change along the way. Traditionally, plan-based models such as the waterfall model have been preferred when developing safety-critical or security-critical systems. This can include systems used in hospitals or in power grids, where bugs and downtime can have a critical impact. In recent years, however, it has been suggested that systems in this category can also be successfully developed using agile methods (Hanssen et al. 2018).

### 2.1.2 Agile software development

The term "agile" in the context of software development has been in use since the late 90s (Dingsøyr, Falessi et al. 2019). Agile is not a specific model or process, but rather a philosophy or set of guidelines. This philosophy was introduced to reduce the overhead found in heavyweight, plan-based methods. In contrast, agile methods are considered lightweight and are characterized by short, iterative development cycles with self-organizing teams (Bourque et al. 2014). In 2001, a group of 17 developers got together and created the values and principles of Agile, which are found in the "Manifesto for Agile Software Development" (Beck, Beedle et al. 2001). The four main values of the manifesto are shown in Figure 2 below and the

twelve principles in Figure 19 in the appendix. The values entail that processes and tools should be adapted to the team members, not the other way around. Working software should be developed early and is more valuable than extensive documentation. Customers, clients, and stakeholders should be involved during the entire development process. As software development can be volatile regarding changes in product requirements, being able to respond to changes is important and should be prioritized over following a plan. This can be done by breaking the project into iterations, where planning is mostly done for one iteration at a time, rather than for the whole project. Planning for only a couple of weeks at a time is simpler as the near future is more certain than the distant future, and less planning is wasted if the project needs to change direction from previous plans. Being agile is something that is easy to do, but hard to master. Therefore, the role of agile coaches has been formed to help guide teams to adopt agile or improve their agile work process. They also have an educational role, teaching teams and stakeholders about agile methods and practices (Daljajev et al. 2020).

As previously mentioned, agile is a philosophy rather than a model. A range of models and processes that claim to be agile exists. Next, three of them will be described, namely *Scrum*, *Kanban*, and *Extreme Programming (XP)*. These three will be presented due to their relevance to the two cases that have been researched. Further, Scrum and Kanban were the two most popular according to the 16th State of Agile Report from Digital.ai (2022), with Scrum and Kanban being employed by 87% and 56% of the respondents respectively.

## Manifesto for Agile Software Development

*We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

**Individuals and interactions** over processes and tools
**Working software** over comprehensive documentation
**Customer collaboration** over contract negotiation
**Responding to change** over following a plan

*That is, while there is value in the items on the right, we value the items on the left more.*

Kent 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, Dave Thomas



Figure 2: Values of the agile manifesto

Source: (Beck, Beedle et al. 2001)

### Scrum

Scrum is one of the most popular methods for agile software development (Sharma and Hasteer 2016). The term "scrum" was first used in the context of product development by Takeuchi and Nonaka (1986), however, the first version of the Scrum framework was presented a decade later in 1995 by Schwaber and Sutherland (2020). A general overview of its life cycle is shown in Figure 3. It is a lightweight, incremental, and iterative framework that aligns well with the agile values and principles. It is also customizable, which makes it a good fit for a lot of different types of development teams. The advised size of a Scrum team is around five to nine people (Kniberg 2015). Further, a Scrum team should be cross-functional, meaning that its members should possess all the skills needed to produce value. A Scrum team

consists of one Scrum Master and one product owner, with the rest of the team members being developers (Schwaber and Sutherland 2020). One important aspect of the Scrum team is that there should be no hierarchy within the team, meaning that all members of the team have the same opportunity to impact decision-making. While the Scrum Master's role may sound like it is above the others, the Scrum Master's role is primarily to ensure that the team follows the framework and philosophy, and does not entail more power within the team. The responsibility of the Product Owner is to maximize the value created by the Scrum team (Schwaber and Sutherland 2020). This can be value in the eyes of the customer, the development organization, or other stakeholders of the project.

The project is broken up into iterations called "sprints" (Schwaber and Sutherland 2020). The team can decide their own sprint length in conjunction with the product owner, with anywhere from one to four weeks being a typical length. The sprint starts with a sprint planning meeting, which is a meeting where the team members, and possibly other stakeholders, plan the current sprint. The highest prioritized product backlog items are put into the sprint backlog, where the team members have to predict how much work each of the items require, and how much the team can get done during the length of the sprint. The planning meeting should also produce a sprint goal that the team can work towards during the sprint. This goal should be testable and clear so that the team can assess whether they met their sprint goal at the end of the sprint. For development teams working full-time on a project, the daily scrum is a daily meeting with the purpose of synchronization within the Scrum team. It is a 15-minute meeting held daily, where developers share what they accomplished the previous day, what they will work on for the day, and whether there are any dependencies with other members or anything else blocking their progress. Although the daily scrum is a short and simple meeting, it can help keep the team aligned and up-to-date with each other, which helps the team stay agile.

At the end of the sprint, the team will typically hold a "demo" or "sprint review". This is a meeting where stakeholders are invited and the team presents the outcome of the sprint and their progress towards the project goals (Schwaber and Sutherland

2020). In addition to this event, another meeting is held to conclude a sprint, called a "sprint retrospective". This is a meeting where the Scrum team looks back on the sprint that has just finished and identifies what the team did well and what could have gone better. The team also select a couple of measures that the team should take during the next sprint, to improve how they work. This event is considered very important, as it allows the team to learn and evolve. Developers adopting Scrum for the first time may feel that there are too many events where the team members are not coding and hence not producing value. However, the reflection that takes place during these meetings tends to help the team improve their work process and in turn, increase productivity.

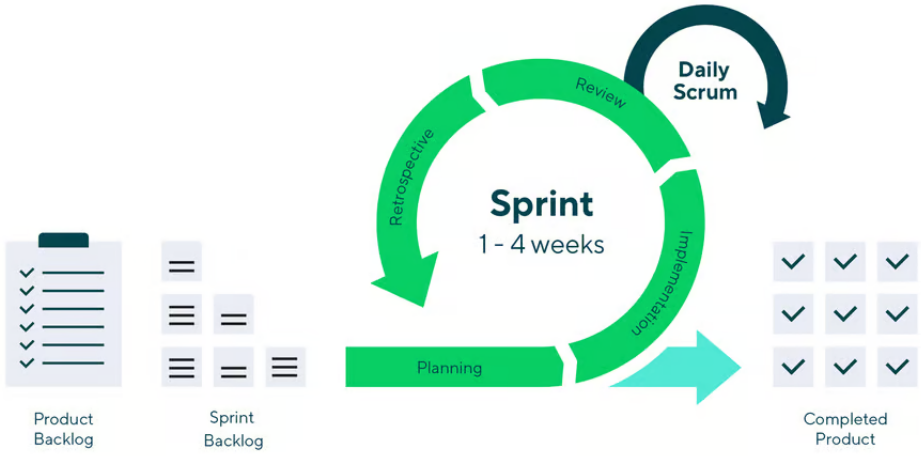



Figure 3: Scrum life cycle

Source: (Shore and Warden 2007)

### Kanban

Kanban is a method of visualizing the progression of work tasks. It was developed as a part of the Toyota Production System and was created to streamline manufacturing at Toyota's factories (Ohno 1988; Sugimori et al. 1977). The practice has since spread to other domains, including software development. To visualize task progression, a Kanban Board is used, which can either be a physical board hanging on the wall of the workspace, or a virtual board living in the digital world. An

example of such a board is shown in Figure 4. No matter what type of board or tool is used to visualize the work progression, it is important that the whole team can access and make changes to the board. The work that a team needs to complete is split into smaller tasks which are hung up on the board. Named columns are used to indicate where each task is in the workflow (Kniberg 2010). Examples of such columns can be "To Do", "In Progress", and "Completed", but the names and number of columns can be adjusted to the team and their context. As work on a task progresses, the task is moved along the board to the column that best describes the state of the task. The board may also include an indication about who is working on a specific task, which can be useful information to the rest of the team. Kanban also suggests limiting the number of tasks that are in progress at a time (Kniberg 2010). Finishing one task is preferred over being in progress of multiple tasks at the same time. Kanban also suggests measuring the average time it takes to finish one task in the team (Kniberg 2010). This can make future work more predictable and improve workload estimates by the team. In general, Kanban is a way of working that is not very prescriptive, meaning that it has few rules and allows for customization. The use of Kanban does not exclude the use of Scrum, in fact, Scrum and Kanban are often used together. The use of these two processes together has been coined "Scrumban" (Nikitina et al. 2012).

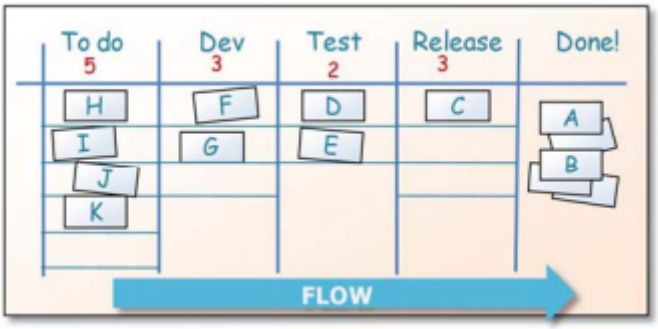



Figure 4: Example of a Kanban Board

Source: (Kniberg 2010)

### Extreme Programming

Extreme Programming, often referred to as "XP", is an agile software development methodology consisting of a range of activities and practices. Its inception is credited

to Kent Beck, who also was among the developers who created the Agile Manifesto (Beck, Beedle et al. 2001). XP utilizes short development iterations or cycles, similar to Scrum. A cycle produces one release, which is some finished part of the product. The iterations should be as short as possible, but the release should still make sense as a whole, meaning a half-finished feature should not be shipped just to make the iteration shorter (Beck and Andres 2004). Development activities such as analysis, design, coding, testing, and deployment can happen simultaneously or within a short time frame, in sharp contrast to plan-based methods (Shore and Warden 2007). It is a more prescriptive framework than Scrum, as it includes most of the principles of Scrum in addition to a set of specific engineering practices (Kniberg 2010).

One of these specific practices is pair programming. XP encourages the use of pair programming where two developers sit together and collaborate to write code with one keyboard and mouse. One developer writes the code, while the other observes and provides suggestions for the developer with the keyboard. Pair programming doubles the amount of available brain power, and allows the person who is not doing the coding to think about strategic and long-term issues or the impact of the code that is written (Shore and Warden 2007). Pair programming is claimed to improve code quality, team focus, and knowledge sharing (Kniberg 2015).

Another practice within XP is collective ownership. This practice suggests that all team members should be able to change any code in the system, not just the parts that they wrote. This does not mean that everyone needs to understand every part of the code base equally well, but everyone should know something about every part (Beck and Andres 2004). One strategy to achieve a high level of collective ownership is to use pair programming with frequent rotation of partners (Kniberg 2015). Using this approach, at least two people will have a good understanding of the code that is written. Enforcing peer reviews on new code changes also boosts the common understanding of the code, as the reviewer will need to read and understand the code written by others, before being able to review it.

A third practice of XP is continuous integration. Integrating small changes at more frequent intervals makes it easier to spot mistakes and limits conflicts with existing code, which in turn can give the developers more time to produce new features and

reduce the time spent on resolving complex merge conflicts. Some suggest that code should be integrated every few hours, or at least once per day for full-time developers (Beck and Andres 2004; Shore and Warden 2007). Continuous integration also implies the use of automated testing, which enforces that the tests are actually run before accepting new changes, potentially catching bugs in the process. In addition to these three practices, XP consists of a variety of other engineering practices. Nine other XP practices follows here, gathered from the book "Extreme Programming Explained" by Beck and Andres (2004):

- **The Planning Game** - Used to determine the scope of the next iteration by estimating workload of tasks.
- **Small releases** - Release a simple system to production quickly, then release new versions on a short cycle.
- **Metaphor** - Guide the development with a simple, shared vision of how the system works.
- **Simple design** - The system should be developed as simply as possible.
- **Testing** - Tests must run flawlessly before the development can continue.
- **Refactoring** - Removes duplicated code, simplifying the system and making it more robust and flexible.
- **40-hour week** - Work no more than 40 hours per week as a general rule.
- **On-site customer** - Include a real user of the system on the team, available to answer questions.
- **Coding standards** - Code should adhere to specific rules and standards set by the team.

### 2.1.3 Lean software development

Lean is a way of working that originates from the Japanese car manufacturer Toyota (Henrik Kniberg 2011). Lean encompasses the Toyota Production System, that

helped make Toyota the most successful car manufacturer in the world. Later, it has been found that the principles used at Toyota are applicable in an array of fields, including software development. The term "lean" was first used in the context of software development in the book "Lean Software Development: An Agile Toolkit" by M. Poppendieck and T. Poppendieck (2003). In this book, the lean methodology is described by presenting seven principles, which will be discussed below.

**Eliminate waste**

Waste is described as anything that does not create value for a customer. This can be a waste of the employees' time, waste of unused inventory, writing documentation that is not used, and so on. Eliminating waste is seen as the most fundamental principle of lean. Therefore, the first step in adopting lean development principles is to identify waste. Some of the sources of waste from the manufacturing world can be directly translated to the software development field, but there are also some differences. To help software development managers to discover waste, M. Poppendieck and T. Poppendieck (2003) has created a list of common waste sources in software development:

- **Partially done work** - Software that is only partially completed ties up resources and can block progress in other units. Software units do not provide direct value until they are integrated into the system.
- **Extra processes** - Paperwork consumes resources that could have been used to create value. Some systems need extensive documentation due to safety-critical requirements, but the team should always search for the most effective way of conveying information.
- **Extra features** - Every unit of software adds complexity to a system, and adds a potential point of failure. Premature implementation of features may lead to the feature becoming obsolete before it is ever used.
- **Task switching** - Context switching consumes time, and is therefore a waste. Belonging to multiple teams increases the amount of task switching and interruptions, and should therefore be avoided.

- **Waiting** - Delays in the development process make team members spend more time waiting and less time creating value. This can include delays in starting a project, delays due to excessive documentation, or delays in approval, integration and deployment of new code.
- **Motion** - The harder it is for developers to get answers to questions that come up, the more time it takes before they can go back to being productive. Making information more available can decrease the time wasted searching for answers.
- **Defects** - Time spent researching and fixing problems in the product are forms of waste. This can be limited by testing, integrating, and releasing to production often.

### Amplify learning

Working in an iterative way has been found to be an effective way of generating knowledge, especially for problems that are hard to define or where the answer is not clear (M. Poppendieck and T. Poppendieck 2003). Software development inherently contains a fair amount of uncertainties, such as what to build and how to do it, and these problems are easier dealt with using short iterations that promote continuous feedback. Even traditional software development models such as the waterfall model was designed with feedback in mind. In practice, however, it does not promote feedback as it assumes that all the details of a project are determined at the beginning, restricting deviations from the initial plan.

### Decide as late as possible

The reasoning behind opting to make decisions as late as possible is that it allows for a more well-informed decision as you obtain more information over time. Following an adaptive rather than a predictive process allows a team to have multiple options open, rather than making commitments early in the project. Predictive processes are well suited for highly predictable situations, as thorough planning can decrease time spent on implementation. However, developing software often involves uncertainties, both technical and domain-related.

**Deliver as fast as possible**

Fast delivery of the software is appreciated by customers and increases business flexibility. It allows companies to have fewer resources tied up due to work in progress, which decreases costs. This principle also complements the principle *decide as late as possible*. If delivery is fast, you can delay decisions longer, which as mentioned entails keeping options open longer and making more well-informed decisions.

**Empower the team**

Allowing teams to be self-organized is encouraged in lean development. This can give individuals space to grow, and make the team more motivated to accomplish its goals as it increases the sense of ownership of the project. Respect and trust are also central aspects of team empowerment, which involves listening and acknowledging team members, and trusting them to make decisions and take risks.

**Build integrity in**

Integrity can be broken up into two dimensions, *perceived integrity* and *conceptual integrity*. Perceived integrity involves that the product has a balance of function, usability, reliability, and economy that satisfies the customer. Communication between customers and developers is an important factor in maintaining this type of integrity. Conceptual integrity, on the other hand, means that the concepts of the system work together in a smooth and cohesive way. To maintain conceptual integrity, refactoring and testing should be used, to keep the code simple and clear and to detect problems early.

**See the whole**

A system is more than the sum of its parts, and simply putting together the best parts may not lead to the best overall system. One trap that some organizations fall for is the exaggerated use of suboptimizations (M. Poppendieck and T. Poppendieck 2003). As systems grow complex, it can be tempting to divide them into more manageable parts that can be dealt with locally. However, while local optimizations can increase performance locally, they tend to decrease overall performance across the organization or project.

### 2.1.4 Large-scale agile software development

In the past, agile methods have been described to be best suited for specific circumstances. That is for small, co-located software development teams (Dingsøyr and N. B. Moe 2013). In 2002, Williams and Cockburn (2003) stated that agile methods were best suited for co-located teams of 50 or fewer people. Since then, however, agile methods have been used increasingly in contexts straying further from where they were intended. One such context is large projects. Implementing agile methods is more challenging in the large-scale context, than for small, less complex projects (Dybå and Dingsøyr 2009). However, agile methods tend to outperform non-agile ones, even for large projects (Jørgensen 2018).

Several definitions for what constitutes large in "large-scale agile" exists. Some of these include the number of people, the number of lines of code, the number of sites, or "when you don't know everyone else working on the same project/product" (Dingsøyr and N. B. Moe 2014). However, for this research the definition from the study "What is Large in Large-Scale? *A Taxonomy of Scale for Agile Software Development*" will be used (Dingsøyr, Fægri et al. 2014). The aim of that study was to develop a taxonomy to define the different levels of scale for agile development. This taxonomy is shown in Table 1. The taxonomy defines size depending on the number of teams in the project. It suggests that agile software development projects with 2-9 teams should be considered "large-scale", while projects with more than 10 teams are categorized as "very large-scale".

Large-scale agile software development projects involve more developers, teams, and stakeholders than smaller projects. With more people involved, it is reasonable to believe that coordination and communication become more challenging. Any individual in the project may not know all the other members of the project or who is responsible for a specific part of the product. When a project member experience troubles or have questions, they may have to go through several people to get to the person who has the answer. This increases the need for knowledge-sharing mechanisms, as well as structures and processes that allow the teams within the project to stay aligned and communicate effectively. To tackle these problems,

a range of various large-scale agile software development frameworks have been created.

Table 1: Scale taxonomy agile development

| Level | Number of teams | Coordination approaches |
|---|---|---|
| Small-scale | 1 | Coordinating the team can be done using agile practices such as daily meetings, common planning, review and retrospective meetings. |
| Large-scale | 2-9 | Coordination of teams can be achieved in a new forum such as a Scrum of Scrums forum. |
| Very large-scale | 10+ | Several forums are needed for coordination, such as multiple Scrum of Scrums. |

Source: (Dingsøyr, Fægri et al. 2014)

### Challenges

Recently, Edison et al. (2022) performed a systematic literature review identifying challenges and success factors in large-scale agile software development from 191 primary studies. The main categories of challenges found were:

- **Inter-team Coordination:** Synchronization and alignment were found to be challenging across dynamic and fast-moving teams. Also, reducing the number of dependencies on the inter-team level could be difficult.
- **Organisational Structure Challenges:** To stay agile, teams should be cross-functional. However, this may be difficult to achieve when domain-specific expertise and specialists are required. Further, roles and responsibilities introduced along with agile methods may not always be easy to assign and can lead to complex organizational setups.
- **Architectural Challenges:** An inability to see the big picture was found to be a challenge. This created friction in relation to handovers between teams.

- **Requirements Engineering Challenges:** Insufficient requirement planning lead to inter-team dependencies. This can be caused by managers lacking knowledge on software development.

- **Customer Collaboration Challenges:** While agile principles promote heavy customer involvement, the management of the relationship between the project and client can be challenging. Business unit managers are generally less familiar with agile methods, leading to friction.

- **Method Adoption Related Challenges:** Adopting complex large-scale agile frameworks with roles and practices can distract companies from achieving their business goals. Scaling agile practices to non-development units is also a common challenge.

- **Change Management Challenges:** A lack of the right mindset can prohibit organizations from gaining all the potential benefits of agile at scale. The organization as a whole needs to be ready to cope with constant, concurrent changes.

- **Team-related Challenges:** A lack of autonomy may be experienced by teams when scaled agile methods are introduced. Additionally, some teams may feel a lack of ownership of tasks and user stories in scaled agile programs, compared to smaller agile projects.

- **Project Management Challenges:** A lack of alignment between agile methods and existing processes on the project management level can cause challenges regarding long-term planning. Further, a lack of meaningful metrics to measure improvements from the adoption of large-scale methods may impact belief in the changes.

#### Success factors

In the same study Edison et al. (2022) found the following main categories of success factors to the adoption of agile at scale:

- **Management and Organizational:** Leadership visibility and support were

important to ensure the success of method adoption. Also, the conviction of stakeholders regarding the agile methods may be essential.

- **Process:** Keeping the development process transparent reduces dependencies and increases coordination between teams. Using communication arenas such as Scrum of Scrums, wikis, and demos also aids coordination.

- **People:** Training and coaching can be essential for the adoption of certain large-scale methods. Sharing knowledge and employing communities of practice can also help with coordination and lead to continuous improvement.

- **Technology:** Technical infrastructure, including joint development tools, test environments, continuous integration, and automated tests, is needed to enable end-to-end development. Adequate infrastructure is also needed to support communication, knowledge sharing, and communities of practice.

#### Large-scale agile software development frameworks

Some examples of large-scale agile development frameworks are Scaled Agile Framework (SAFe), Large Scale Scrum (LeSS), the Spotify model, Scrum-at-Scale, and Nexus. These frameworks incorporate workflow patterns and routines and are supported by a set of tools (Conboy and Carroll 2019). Due to its relevance to one of the cases of this study, the Spotify model will be described further.

**The Spotify model** is a framework that encompasses the way Spotify has scaled agile methods, being a rapidly growing tech organization. It was introduced in 2012 and has since become a popular model in the world of agile transformations. While other scaling frameworks focus on specific practices, the Spotify model deals with how an organization can be structured to enable agility and promotes team autonomy in terms of selecting specific practices[5]. An overview of how teams and people are organized when using the Spotify model is shown in Figure 5. The organizational structure is made up of the following components, as described by Kniberg and Ivarsson (2012):

[5]https://www.atlassian.com/agile/agile-at-scale/spotify

- **Squads:** The smallest organizational unit, they are a cross-functional, self-organizing type of team who decide their own way of working. Each squad is responsible for its own part of the total system and has its own long-term mission.

- **Tribes:** A collection of squads that work in related areas. Each tribe has a tribe lead who is responsible for facilitating and enabling the best possible development environment for the squads. Tribes are designed to be smaller than about 100 people, due to the limiting factor of how many people humans are able to maintain a social relationship with.

- **Chapters:** A small, inter-squad group of people within the same tribe who have similar skills and work with similar types of tasks. The chapters meet regularly to discuss their area of expertise and specific challenges. This helps prevent autonomous squads from being completely isolated and allows for solutions and knowledge to be shared across squads.

- **Guilds:** A more wide-reaching "community of interest" which reaches across tribes. Guilds consist of a group of people who want to share knowledge, tools, code, and practices. It is a more organic structure, as anyone with an interest can join, whether they work within the topic of discussion or not.

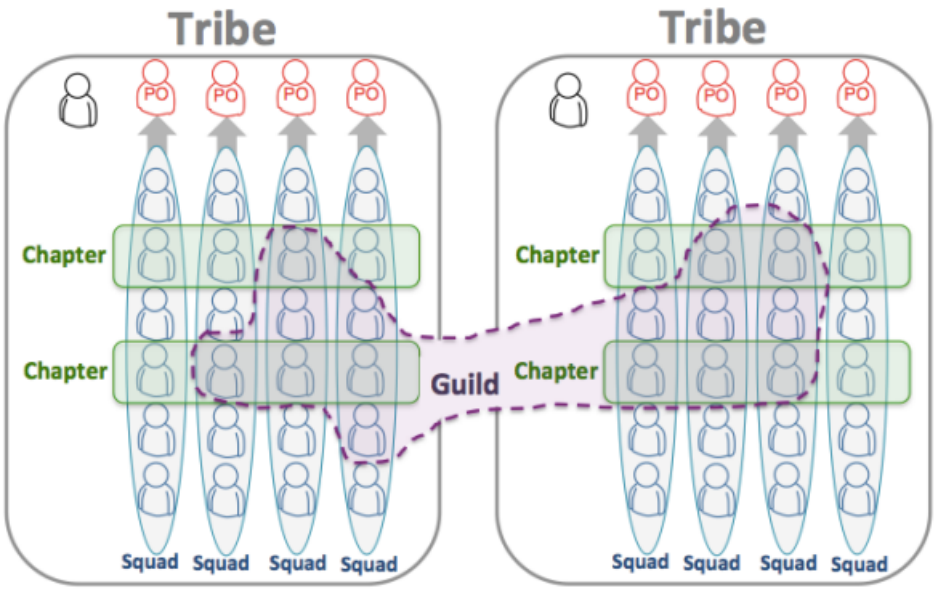



Figure 5: Organization structure using the Spotify model

Source: (Kniberg and Ivarsson 2012)

### Scrum of Scrums

A commonly used forum in several of the large-scale agile frameworks is the Scrum of Scrums. This is an arena that is used for the coordination of multiple agile teams within a project (Schwaber 2004). One member from each of the teams is chosen to participate in the Scrum of Scrums meeting. This meeting will consist of the representatives from each team sharing the progress of their teams, what the team will work on going forward, and if there is anything blocking them from progressing. This will give the representatives valuable insight into the other teams. It will also allow the representatives to identify dependencies between the teams, and coordinate their work accordingly to resolve these dependencies. For very large projects, there may even be a need for an additional layer to this structure, with a forum called Scrum of Scrums of Scrums[6]. Similarly to the Scrum of Scrums, this meeting consists of one representative from each unit of the layer below, namely the Scrum of Scrums. An illustration of this structure can be seen in Figure 6.

[6]https://www.agilest.org/scaled-agile/scrum-of-scrums/

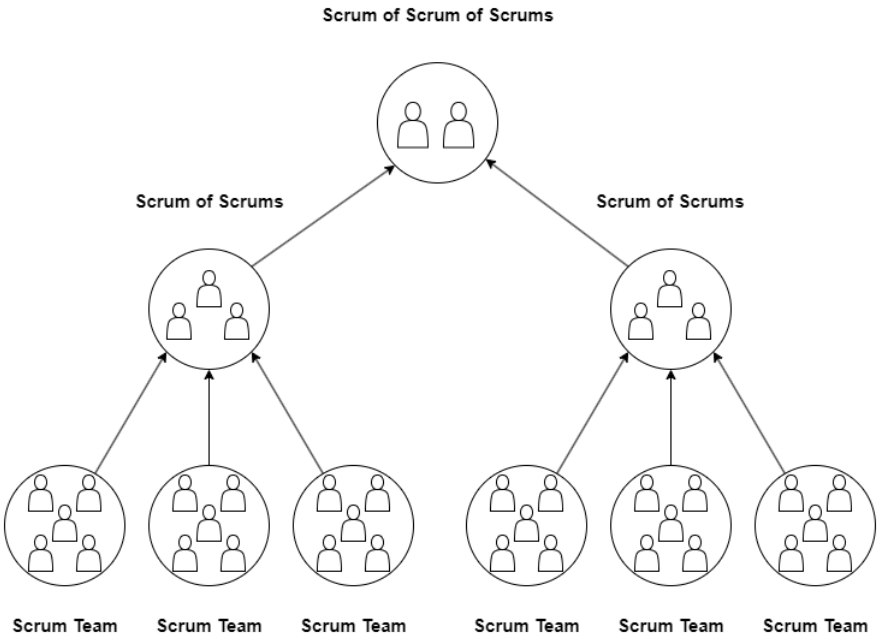



Figure 6: Scrum of Scrums of Scrums, illustration inspired by: (*Scrum of scrums - guide to agile scaling frameworks* 2016)

## 2.2 Teams and teamwork

Teams and teamwork are fundamental aspects of many facets of society. They can be found in areas such as sports, voluntary organizations, school projects, and the workplace. Companies in a wide range of industries rely on people working together and cooperating in order to achieve their goals and produce value for their customers. In this section, existing research on the topic of teams and teamwork will be presented.

### 2.2.1 Teams

Teams are found everywhere, and it is reasonable to believe that most people have a sound notion of what a team is. However, a concrete definition of the term can be beneficial when studying teams as a concept. Kozlowski and Bell (2003) define teams as follows:

(a) are composed of two or more individuals

(b) who exist to perform organizationally relevant tasks

(c) share one or more common goals

(d) exhibit task interdependencies (i.e., workflow, goals, knowledge, and outcomes)

(e) interact socially (face-to-face or, increasingly, virtually)

(f) maintain and manage boundaries

(g) are embedded in an organizational context that sets boundaries, constrains the team, and influences exchanges with other units in the broader entity

All the teams participating in this study fulfill each of these seven requirements.

### 2.2.2 Team autonomy

Team autonomy is a central part of agile, and one of the twelve principles from the agile manifesto states that *"the best architectures, requirements, and designs emerge from self-organizing teams"* (Beck, Beedle et al. 2001). Autonomy has been defined as the amount of freedom an individual has in carrying out assigned tasks (Langfred 2007). Langfred further explains that team-level autonomy is the amount of freedom a team has in carrying out tasks within its organization, and that the label "self-managing" is generally given to teams with a high degree of team-level autonomy. Autonomous work groups have been found to outperform traditional groups, however, a high degree of autonomy may introduce difficulties with coordination across teams (Ingvaldsen and Rolfsen 2012). This is a highly relevant challenge in large-scale agile, where there is a need for both inter-team coordination and team autonomy (N. Moe et al. 2016). Further, it has been found that autonomy positively influences team performance when tasks are highly independent, while the impact is negative when task interdependence is low (Stray et al. 2018).

### 2.2.3 Collaboration

Working together by collaborating is something that is needed in many fields of work, and software development is no different. Collaboration has been defined as:

*"collaboration takes place when two or more people are working together on a task"* (Sharp and Robinson 2010). As software development can be a complex matter, joint effort from several developers is usually required, which creates a need for collaboration. Successful collaboration is characterized by the presence of a specific outcome, such as a product or desired performance, which is achieved through group effort (Kotlarsky and Oshri 2005). In the study of Sharp and Robinson (2010), the prevalence and importance of story cards in agile teams were highlighted. Such cards could be used to record user stories, estimates, tests, or rough designs and ideas. Story cards are hung up on a wall that the whole team can see, and tasks can be picked directly from the wall. The story cards and wall aids communication about tasks, progress, and ideas within the team, which can improve collaboration. Further, Sharp and Robinson (2010) found that the following collaboration mechanisms were used to aid teamwork in agile teams:

- **Story cards** - Promotes collaboration between developers and customers. They also generate activity and information exchange and support knowledge sharing, which in turn helps team members collaborate more effectively.
- **The Wall** - Is available to view at all times and is regularly updated. Contains immediately relevant information and makes collaboration during stand-ups and pairing more effective.
- **Pair programming** - Pairs are swapped regularly which contributes to knowledge-sharing, making collaboration easier.
- **Colocation** - When team members sit in an open plan environment, they can overhear problems, questions, solutions, and discussions from others, and they can tune in to the conversations that are relevant or of interest.
- **Stand-up meeting** - Promotes knowledge sharing and encourages everyone to contribute to ongoing tasks, leading to a collective approach to problem-solving.

### 2.2.4 Teamwork in general

What is teamwork, and how can it be measured? These are some of the questions that Hoegl et al. (2001) attempt to answer in their study on teamwork in the context of innovative projects. In this research, a concept of team collaboration is developed, named Teamwork Quality (TWQ). This model consists of six components, which are shown in Table 2. Drawing on previous research, Hoegl et al. divide project success into two categories, *team performance* and *personal success*. They define team performance as the extent to which a team meets quality, cost, and time objectives. Personal success, on the other hand, involves increasing team members' motivation, as well as personal satisfaction and learning. The central proposition of the study by Hoegl et al. is that TWQ is positively related to the success of innovative projects. This involves that a higher degree of teamwork quality increases both team performance and personal success.

Table 2: The Teamwork Quality Construct

| | |
|---|---|
| Communication | Is there sufficiently frequent, informal, direct, and open communication? |
| Coordination | Are there individual efforts well structured and synchronized within the team? |
| Balance of Member Contributions | Are all team members able to bring in their expertise to their full potential? |
| Mutual Support | Do team members help and support each other in carrying out their tasks? |
| Effort | Do team members exert all efforts to the team's tasks? |
| Cohesion | Are team members motivated to maintain the team? Is there team spirit? |

Source: (Hoegl et al. 2001)

There are many ways to work together as a team, but increasing the effectiveness of teamwork can be a good step towards improving productivity in a project. First,

a definition of teamwork effectiveness is needed. Salas et al. (2005) distinguish between the terms "team performance" and "team effectiveness". They describe that team performance solely accounts for the outcomes of a team's actions, regardless of how tasks were accomplished. Team effectiveness, however, does not only consider whether the team performed but also how the team interacted to achieve its outcome. To improve teamwork effectiveness, it may be helpful to first get an indication of which factors impact teamwork. In "Is there a "Big Five" in Teamwork?", Salas et al. (2005) argue that it is possible to group what researchers know about teamwork into five core components. These five components are: *team leadership*, *mutual performance monitoring*, *backup behavior*, *adaptability*, and *team orientation*. Additionally, the researchers identified that these components are supported by coordinating mechanisms, such as *shared mental models*, *closed-loop communication*, and *mutual trust*. These coordinating mechanisms are required to extract value from each of the core components. This resulted in the Big Five model, which is illustrated in Figure 7. The definition of each of the core components and coordinating mechanisms are presented in Table 3 and Table 4.

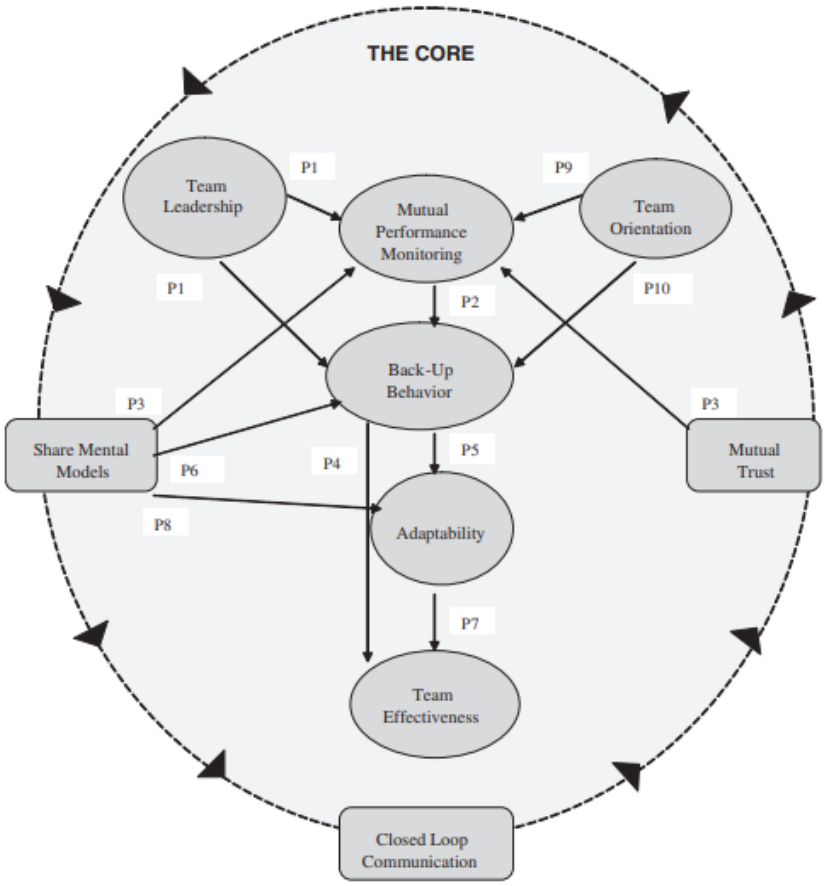



Figure 7: The Big Five model

Source: (Salas et al. 2005)

Table 3: Definitions of the five core components in the Big Five Model

| **Core component** | **Definition** |
|---|---|
| Team leadership | Ability to direct and coordinate the activities of other team members, assess team effectiveness, assign tasks, develop team knowledge, skills, and abilities, motivate team members, plan and organize, and establish a positive atmosphere. |
| Mutual performance monitoring | The ability to develop common understandings of the team environment and apply appropriate task strategies to accurately monitor teammate performance. |
| Backup behaviour | Ability to anticipate other team members' needs through accurate knowledge about their responsibilities. This includes the ability to shift workload among members to achieve balance during high periods of workload or pressure. |
| Adaptability | Ability to adjust strategies based on information gathered from the environment through the use of backup behaviour and reallocation of intra-team resources. Altering a course of action or team repertoire in response to changing conditions (internal or external). |
| Team orientation | Propensity to take other's behaviour into account during group interaction and the belief in the importance of team goal's over individual members' goals. |

Source: (Salas et al. 2005)

Table 4: Definitions of the three coordinating mechansims in the Big Five Model

| Coordinating mechanism | Definition |
|---|---|
| Shared mental models | An organizing knowledge structure of the relationships among the task the team is engaged in and how the team members will interact. |
| Mutual trust | The shared belief that team members will perform their roles and protect the interests of their teammates. |
| Closed-loop communication | The exchange of information between a sender and a receiver irrespective of the medium. |

Source: (Salas et al. 2005)

### 2.2.5 Teamwork in software development

Being able to collaborate with others is a skill in a lot of different industries, including software development. Developing complex systems is too much work to be completed by one developer in a reasonable period of time. Therefore, the work needs to be split into tasks that can be distributed among the developers of a product. In a perfect world, these tasks would be so well defined that the developers could work in isolation on their tasks without interacting with others. However, this is not reasonable in a real-world scenario, which introduces the need for teamwork, collaboration, coordination, and communication. But is teamwork in software development different from teamwork in other fields? This is discussed by Dingsøyr and Dybå (2012) when they asses whether separate team effectiveness models are needed for the software development field. They list five main issues that should be prioritized in future studies on software team effectiveness, which are the following:

- Better Measurement
- More Rigorous Industrial Case Studies
- Better Understanding of Dynamic Configurations

- Increased Emphasis on Team Cognition
- Better Understanding of Multicultural Contexts

Conclusively, Dingsøyr and Dybå (2012) call for more empirical studies, as well as better theoretical grounding in studies of software team effectiveness. They also express a need for testing and potentially adjusting existing theories from other fields in the context of software development teams. This is a part of the gap that this research project aims to fill. This research will provide an empirical study on the teamwork of software development teams, specifically in a large-scale agile context. It will also compare findings to an existing model, to investigate if the model is applicable in an additional context.

### 2.2.6 Teamwork in agile software development

One of the tenets of the agile manifesto is "Individuals and interactions over processes and tools" (Beck, Beedle et al. 2001). This indicates that being able to work together with others is central to agile development. Agile methods generally promote working in iterations where up-front planning may be limited and short-term planning is preferred over long-term planning. This allows teams to be able to respond quickly to changes but may introduce other challenges. Thorough planning before starting implementation could give a more complete overview of the entire system that is to be developed, which could have allowed for dependencies between developers, teams, and components to be identified. This may suggest that effective teamwork, communication, and coordination are even more important when working in an agile way.

A teamwork effectiveness model (ATEM) is a teamwork effectiveness model developed for agile software development (D. Strode, Dingsøyr et al. 2022). This model is based on the Big Five model which has been discussed previously, but is specifically adjusted for agile development. They collected data from focus groups and case studies to investigate what practitioners meant impacted teamwork effectiveness. Both items that foster and items that hinder team effectiveness were

identified. While some of the core components from the Big Five model were supported, others were changed to better suit teamwork in this context. *Team leadership* was changed to *shared leadership*, as their research suggested that in agile teams leadership is often shared across the members of a team, rather than having a single acknowledged leader. *Mutual performance monitoring* was changed to *peer feedback* as it was believed that this would be a more understandable term for agile teams, and to avoid the negative connotations that the word "monitoring" may bring. *Backup behaviour* was changed to *redundancy* as a redundancy of skills is needed in agile development to enable backup behavior. They also argued that *redundancy* is a more familiar term to software developers. Additionally, one of the coordinating mechanisms was altered. *Closed-loop communication* was changed to *communication* as it was found that in agile development, communication often occurred in groups or to the whole team, rather than one-to-one conversations (D. Strode, Dingsøyr et al. 2022). An illustration of the resulting model is shown in Figure 8 where the five core components are displayed in the middle, and the three coordinating mechanisms on the perimeter.

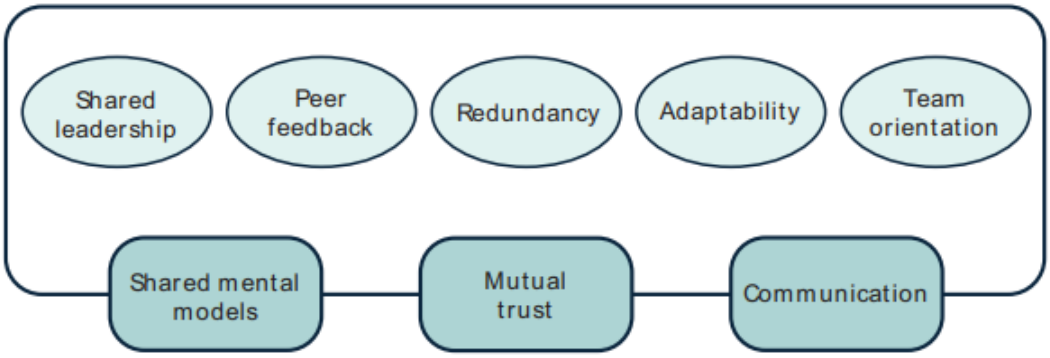



Figure 8: A teamwork effectiveness model (ATEM)

Source: (D. Strode, Dingsøyr et al. 2022)

## 2.3 Coordination

Most people have a general intuition of what coordination is, however, it is a term that has been defined in various ways. This section will aim to explain what coordination is, and why it is important.

### 2.3.1 Coordination in general

When working together on a project, coordination is needed to make sure that everyone involved is aligned with each other. In order to have successful teamwork, a team also needs to have sufficient coordination. One definition of the term "coordination" can be found in the study of Malone and Crowston (1994), "The interdisciplinary study of coordination". In this paper the following definition is used: *"Coordination is managing dependencies between activities"*. This suggests that coordination and dependencies are strongly linked together and that the need for coordination arises when dependencies are identified. Being able to handle and resolve dependencies is important to maintain the progression of work within a team, by trying to minimize the amount of time the team is blocked from progressing.

Another definition of "coordination" is found in "Determinants of Coordination Modes Within Organizations", where the term is described as: *"integrating or linking together different parts of an organization to accomplish a collective set of tasks"* (Ven et al. 1976). In other words, coordination is needed to finish tasks that require effort by multiple people, teams, or other substructures of an organization. By maintaining effective coordination, organizations can keep productivity high to continuously provide value to their customers, clients, or users. In their paper, Van de Ven et al. propose that mechanisms for coordinating work can be grouped into three different categories: *impersonal*, *personal*, and *group* modes. They also build upon the work of March and Simon (1958) who suggest that coordination of organizations can be accomplished in two general ways, *by programming* or *by feedback*. Coordination by programming is coordination that follows standardized information and communication systems. This includes mechanisms such as plans, schedules, rules, and policies, among others. When such mechanisms are implemented, minimal verbal communication between organization members is needed to coordinate work. Ven et al. (1976) classify coordination by programming as an impersonal coordination mode. On the other hand, coordination by feedback has been defined as mutual adjustments based on new information (Thompson 1967). This type of coordination can both be found in personal and group modes.

- **Personal mode** - Mutual task adjustments can be made through either vertical or horizontal channels of communication. Vertical channels tend to go through managers or supervisors. In contrast, horizontal communication consists of one-to-one dialogues between organization members in a non-hierarchical environment.

- **Group mode** - Group mode coordination consists of scheduled and unscheduled meetings where organization members are involved. Scheduled meetings are used for formal, planned, routine communication, while unscheduled meetings represent ad-hoc discussions of work-related problems with more than two participants.

- **Impersonal mode** - The impersonal mode of coordination represents mechanisms and communication that tend to be in written form, rather than requiring verbal communication.

### 2.3.2 Coordination in software development

Coordination is needed in a wide range of industries, and software development is no exception. In this industry, coordination implies that people working on a project agrees upon what they are building, share information with each other, and plan out how and when to accomplish the required activities. The project members should have a shared, common view of what the software should do, how it should be organized, and how it should be intertwined with existing systems. It has been argued that coordination becomes much more difficult, as the size and complexity of a project increases (Kraut and Streeter 1995). In a world that is becoming increasingly digital, firms within a range of industries encounter numerous challenges related to digitization. These include changes to requirements, markets, and regulations, as well as the ever-changing technological evolution (Fuchs and Hess 2018). In order to tackle these challenges, organizations need to be well-coordinated to stay on top of uncontrollable changes.

### 2.3.3 Coordination in agile software development

A popular way of dealing with issues related to digital transformations is the introduction of agile methods (Fuchs and Hess 2018). Working in an agile way can help with dealing with uncertainties, as less planning is done up-front, and implementation of the product starts early in the project. This involves that decisions are easier to reverse, and allows the project to change course with fewer consequences. However, less thorough planning and documentation may lead to other challenges. As less time is spent on planning ahead of implementation, more dependencies between people, teams, or components may be left unidentified until they are discovered during implementation. This might increase the need for coordination, especially in the personal and group modes, in the lack of well-documented plans.

Some research on how agile software development projects achieve effective coordination has been conducted, such as the model developed by D. E. Strode et al. (2012). In this research, practices that are used to coordinate work within the project are defined as coordination mechanisms. In combination with each other, these coordination mechanisms form a coordination strategy. A coordination strategy in the relevant context consists of three components: *synchronization*, *structure*, and *boundary spanning*. In addition, a concept of coordination effectiveness is developed, which consists of *implicit* and *explicit* coordination. Further, it is suggested that a coordination strategy increases coordination effectiveness in an agile context, which makes the model applicable to practitioners who aim to improve coordination in their projects. This model is depicted in Figure 9.

#### Syncronization

Synchronization is something that is achieved by making use of synchronization activities and artifacts (D. E. Strode et al. 2012). Such activities can take place with different frequencies, either once per project, once per iteration, daily, or ad-hoc. These activities are used to promote a common understanding of tasks, processes, or other team member's expertise. Examples of synchronization activities include sprint planning sessions, daily standups, retrospectives, and product demos. Synchronization artifacts are items that are produced during synchronization activities

that are available to the team. This can be items such as backlogs, user stories, or burn-down charts.

### Structure

Structure in relation to coordination effectiveness is described as the arrangement of and relations between parts of something complex (D. E. Strode et al. 2012). Coordination mechanisms related to structure can be grouped into three categories: *proximity*, *availability*, and *substitutability*. Close proximity of the team members is often included in agile methods and allows for pair programming as well as effective communication (Beck and Andres 2004). Availability refers to team members being available to each other. This is achieved by having team members work full-time on one project. Lastly, substitutability is achieved by having technical redundancy, meaning several people have a similar set of skills and expertise. This allows the project to maintain progress even when some of the team members are unavailable. Technical redundancy is achieved either by knowledge sharing or by hiring personnel with similar skill sets.

### Boundary spanning

Boundary spanning happens when someone from within the project needs to interact with someone from outside the project, either other organizations or other parts of the same organization (D. E. Strode et al. 2012). Boundary spanning is made up of three aspects: *boundary spanning activities*, *the production of boundary spanning artifacts*, and *coordinatior roles*. Boundary-spanning activities occur when different units interact to share expertise. Such activities include workshops, user story prioritization sessions, and formal or informal meetings, where customers or others that are not a part of the project teams are present. Boundary-spanning artifacts are produced to support boundary-spanning activities. They are also used to enable coordination beyond the limits of the team or project. One example of such an artifact is project management plans, which can give insight into project progression to external parties. The coordinator role is also a mechanism to achieve boundary spanning. The person with this role coordinates between the project and those outside the project. This can for example be coordination between project

development teams and higher management units of the organization. Coordination is achieved via communication, as well as the sharing of documents such as reports.

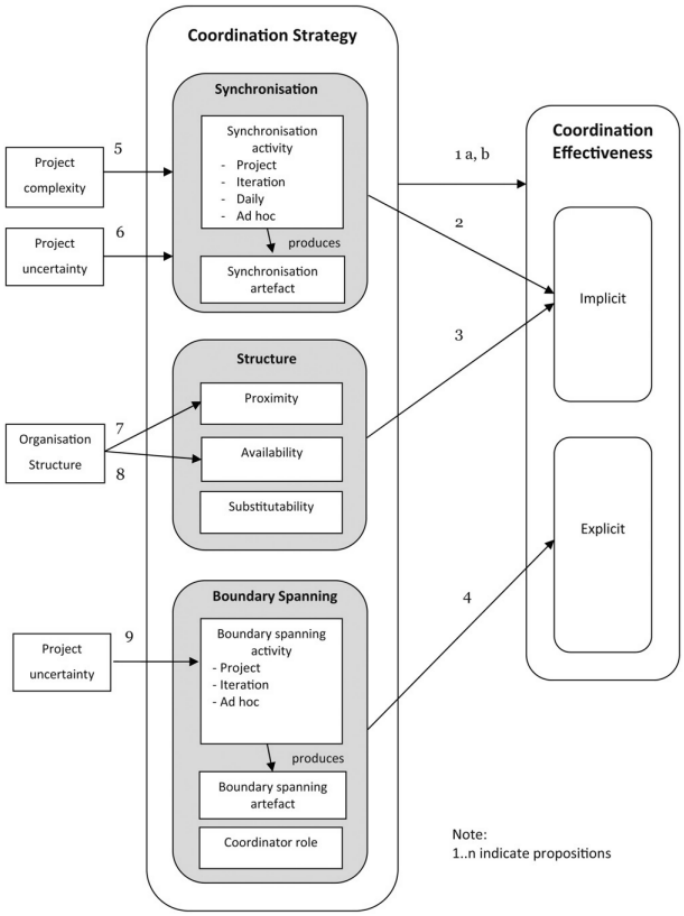



Figure 9: Theoretical model for coordination

Source: (D. E. Strode et al. 2012)

Coordination effectiveness is the outcome of a coordination strategy, or the state of coordination that is achieved in a project. As mentioned, D. E. Strode et al. (2012) suggest that coordination effectiveness is composed of two components, explicit and implicit coordination. Explicit coordination relates to objects, such as the people and artifacts that are involved in a project. Explicit coordination is deemed effective when required objects are in the right place, at the right time, ready, and available to those who need them. Implicit coordination, on the other hand, is coordination that takes place without explicit verbal or written communication. It involves tacit knowledge that the project members have, such as *"Who knows what"* and *"What to do when"*. A model of the concept of coordination effectiveness is shown in Figure 10.

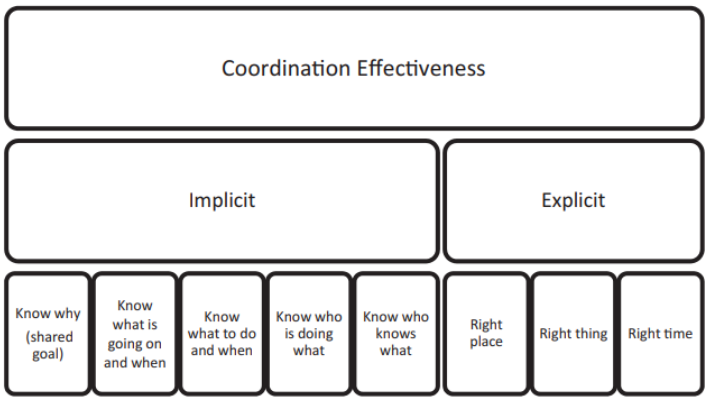



Figure 10: Coordination effectiveness

Source: (D. E. Strode et al. 2012)

As previously discussed, coordination is strongly linked to dependencies. Hence, it is useful to identify what types of dependencies exist in order to solve dependencies and improve coordination. This is the goal of D. Strode and Huff (2012) in their study "A Taxonomy of Dependencies in Agile Software Development". In this research, three distinct coordination categories are identified, which are *task*, *resource*, and *knowledge* dependencies. A task dependency occurs when one task needs to be completed for another task to be able to continue. Resource dependencies occur when an object is needed for the project to progress. Lastly, knowledge dependencies concern dependencies where some form of information is needed in order to continue to progress in the project (D. Strode and Huff 2012). This taxonomy of dependencies in agile software development is depicted in Table 5.

Table 5: Taxonomy of dependencies

| Dependency category | Dependency |
|---|---|
| Knowledge | Requirement |
| | Expertise |
| | Task allocation |
| | Historical |
| Task | Activity |
| | Business process |
| Resource | Entity |
| | Technical |

Source: (D. Strode and Huff 2012)

## 2.4 Remote and hybrid work

In an increasingly digital world, a wide range of jobs can be performed without being in the same physical space as your coworkers. This includes software developers, who after all spend most of their time in front of a computer with an Internet connection. In fact, "ICT professionals" was the profession with the second highest share of employees working remotely in the EU in 2018 (Micaela 2020). One of the agile philosophy principles conveys that face-to-face conversations are the most effective form of communication for development teams (Beck, Beedle et al. 2001). However, working remotely can bring several practical benefits, such as allowing non-colocated teams to collaborate, and expanding the recruitment area for open positions. According to Deshpande et al. (2016), three different types of non-colocated teams exist.

1. **Distributed** teams are teams where sub-teams are located on different sites, often in different countries.
2. **Dispersed** teams are teams where team members work alone from different locations.
3. **Hybrid** teams are teams where some team members work remotely, and others

are colocated.

Working from home can also bring personal benefits to employees. Common examples of this include increased flexibility of schedule, freedom from interruptions, and time saved from not having to commute (DeSanctis 1984). It has been argued, however, that to have success with working from home, a higher level of self-motivation and self-discipline is required (Olson 1983). Additionally, some cited drawbacks of working remotely are isolation, both socially and professionally, as well as difficulties separating home affairs from professional ones (Bailey and Kurland 2002; Klopotek 2017).

While remote work is nothing new, its prevalence rose to new highs during the Covid-19 pandemic. In the EU, the share of workers who usually worked from home was about 5.4% in the time period between 2009 and 2019. During the pandemic, however, the share of workers who worked from home full-time rose to around 40% (Micaela 2020). Even though drawbacks exist, the sudden shift toward remote work has made several organizations realize its benefits, with opportunities for hybrid work likely to remain as a permanent option for many (Phillips 2020). This notion is consistent with the results of the 15th State of Agile Report, where only 3% of the respondents who had worked remotely during the pandemic planned on returning to the office full time (Digital.ai 2021).

## 2.5 Knowledge management

Several different definitions of knowledge management exist, one of these states that: *"knowledge management concerns the formalization of and access to experience, knowledge, and expertise that create new capabilities, enable superior performance, encourage innovation, and enhance customer value"* (Gloet and Terziovski 2004). Developing software is a knowledge-intensive practice. Hence, managing the knowledge that exists within a software development organization by enabling access to experience, knowledge, and expertise may improve performance. The study of Hansen et al. (1999) refers to two main strategies for knowledge management:

- **Codification:** A strategy where knowledge is codified and stored in databases, easily accessible for anyone in the company.
- **Personalization:** A strategy in which knowledge is shared directly from person to person. In this strategy, technology is mainly used for communicating knowledge, rather than storing it.

Further, Earl (2001) has proposed that knowledge management can be separated into seven schools, which can be categorized into three categories: *technocratic*, *economic*, and *behavioral*. These schools are shown in Table 6. The technocratic schools are concerned with the use of technology to manage knowledge. This can include the use of knowledge bases, directories, and shared databases to store, manage, and enable access to useful information. The economic and commercial school looks at knowledge as an asset to the company and is concerned with how knowledge and intellect can be utilized to generate revenue. Lastly, the behavioral schools focus on how management can enable the creation, sharing, and use of knowledge as a resource. This includes creating a sociable culture and encouraging the exchange of knowledge.

In the research of Bjørnson and Dingsøyr (2008) it was found that studies on knowledge management in the software engineering field generally focus on the technocratic schools and management of explicit knowledge. They further suggest that future research should focus more on the organizational school of knowledge management, as organizational activities can be inexpensive to implement and relevant to both agile and traditional software development. Additionally, they propose a stronger focus on the management of tacit knowledge due to the lack thereof in existing studies, and its relevance to agile development.

Table 6: Schools of knowledge management

| Category | School(s) |
|---|---|
| **Technocratic** | Systems, Cartographic, Engineering |
| **Economic** | Commercial |
| **Behavioral** | Organizational, Spatial, Strategic |

Source: (Earl 2001)

# 3 Method

In order to answer the research questions of Section 1.2, a research method was employed. This section will describe each step of the research process, from research design to data analysis. The selected research method will be presented, with arguments for the choices that were made. Criteria that were used in case selection will be shared, in addition to a description of the cases that were chosen. The choice of data generation method will also be presented, along with a description of the analysis phase. Lastly, the section will assess the validity of the chosen research method, and present relevant limitations.

## 3.1 Research strategy and method

The chosen research methodology for this research is a case study approach, more specifically a multi-case study, as two cases are involved. A multi-case study investigates the same phenomena in two or more cases (Yin 2018). In this study, two large-scale agile software development organizations were researched to learn about teamwork effectiveness in this context. The case study approach can give rich insight into one or a few cases, rather than limited insight into a lot of cases. This high level of detail may be needed to be able to determine factors that impact teamwork effectiveness in this context. This approach also allows for a software development team or project to be studied in its natural setting, which can give a more accurate view of the real-life case. Further, the case study approach is well suited for research within the field of software engineering, as the objects of study are hard to study in isolation (Runeson and Höst 2009).

A case study approach can be described as holistic, meaning that relationships and processes and the interconnections between them can be researched, rather than trying to isolate factors (Oates 2006). As teamwork and human interaction are complex matters, the sum of individual factors may not tell the whole story, and studying them in relation to each other may paint a better picture. By using a case study, multiple different data generation methods and sources may be used to collect

data. For this research, interviews were the data generation method of choice, which can yield detailed data and insight into a case. The data from the interviews were analyzed using a qualitative approach. In preparation for this research, a literature review on coordination challenges in large-scale agile software development projects was performed (Woldseth 2022). This initial research revealed several gaps in the research of large-scale agile, and as such, this topic was selected for further research. A visual model of the chosen research process is shown in Figure 11. The research is explanatory in nature, as it will try to identify factors that have an impact on the topic of research, teamwork effectiveness (Oates 2006).

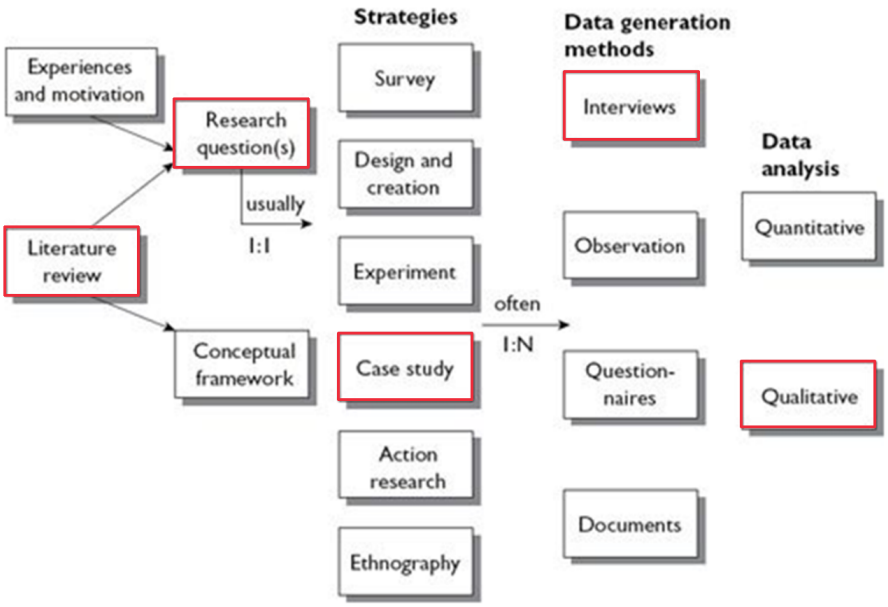



Figure 11: Chosen research process marked in red, template model from Oates (2006)

## 3.2 Case selection

For the case selection, the aim was to find an organization that fits the context of the research. For this study that meant a software development organization that has projects or products where multiple teams collaborate and use agile methods, constituting the use of large-scale agile development. The context of large-scale agile is of specific interest as agile methods were originally made for small, co-located teams, and less research exists on larger projects employing agile methods

(Williams and Cockburn 2003). There are no requirements for the project to use a specific large-scale agile framework, but rather that they use some agile methods and follow an agile philosophy. When defining what constitutes "large-scale agile", the taxonomy from Dingsøyr, Fægri et al. (2014) is used. In this taxonomy, it is proposed that an agile project can be defined as "large-scale" when 2 or more agile teams cooperate. However, while a two-team project would be sufficient to be labeled as "large-scale", the larger the size of the project, the more interesting it is for investigating large-scale effects. The goal is therefore to find an extreme case, in terms of size. Gaining access to such a case may not be easy, therefore the case selection is also bound by the ability to receive access. As such, two Norwegian cases have been selected due to contacts that made it easier to gain access.

### 3.2.1 Case A: Signicat

Signicat is a Norwegian technology company that was established in 2006. They provide online identity services, including electronic identification, authentication, and electronic signature. Signicat has approximately 425 employees, with 175 of those working in tech. A large share of their employees works as developers, while other roles include project managers, economists, lawyers, salespeople, and customer support. They have 10 agile teams, with a structural organization inspired by the large-scale agile framework, the Spotify model. This type of organization was introduced to the company in 2021. This entails that teams within the same feature area are organized in a tribe structure. Additionally, inter-team guild communities are used, where those interested in a topic can come together to share ideas, discuss, and collaborate across tribes. Since its inception, Signicat has grown rapidly and was in 2021 ranked as one of the fastest-growing companies in Europe[7]. Today, they sell their services to more than 6,000 small and large companies, including prominent firms such as Allianz, Volvo, and Sopra Steria, resulting in a revenue of around 620 million NOK in 2021[8]. With headquarters in Trondheim Norway, Signicat has expanded with offices in several other European countries over the past years. Their

[7]https://www.ft.com/content/8b37a92b-15e6-4b9c-8427-315a8b5f4332

[8]https://www.signicat.com/no/pressemeldinger/european-focus-gives-signicat-high-2021-growth-in-revenue-and-ebitda

locations are visualized in Figure 12.


Figure 12: Map of Signicat office locations

### 3.2.2 Case B: NAV IT

The Norwegian Labour and Welfare Administration (NAV) is the public welfare agency of Norway. Founded in 2006, the government agency employs approximately 22,000 people across 264 office locations throughout the country. NAV administers state programs including unemployment benefits, pensions, child benefits, and sick pay, and they administer a third of the Norwegian national budget. To manage all the services that NAV provides, an extensive collection of IT systems is necessary. The end users of these systems are organizations and individuals who apply for benefits, employees in NAV who manage the grants, and external organizations who use the data that is produced. Since its inception, NAV has outsourced the

development and maintenance of its IT systems to externals. However, in 2017, they decided to build up their own IT department and move the development and maintenance of most of their IT systems in-house (Mohagheghi and Lassenius 2021). During this transition, it was also decided to adopt agile methods, which was found to be a success factor in other public software projects (Mohagheghi and Jørgensen 2017). Today, NAV IT consists of approximately 800 employees and has an annual budget of about 1.3 billion NOK. This includes about 30 architects, 80 designers, more than 300 developers, 180 technicians (operations and infrastructure), and other leadership, advisor, and support roles. The members of NAV IT are organized into around 80 cross-functional teams and their organization is inspired by the "Team Topologies" approach (Skelton and Pais 2019). This includes using team types such as platform teams, enabling teams, and support teams. Their development is also organized in eight product areas, where one of these, the pension product area, was studied in detail in this study through interviews. This product area ensures that more than a million pensioners get their retirement pension, and is responsible for the services that manage 75% NAV's entire benefits budget.

These two cases are relevant to this study as they conduct agile software development and belong to the large-scale context. Although this is a similarity between the two, they are also different in some aspects. Some of these are displayed in the overview of the two cases found in Table 7. During this study, eight interviews were conducted with participants from Signicat, while six interviews were conducted with participants from NAV IT.

Table 7: Overview of the two cases

| | **Case A: Signicat** | **Case B: NAV IT** |
|---|---|---|
| Industry | Online identity services | Government grants and benefits |
| Economy | NOK 620 M (revenue 2021) | NOK 1.3 Bn (budget) |
| Employees | approx. 425 | approx. 800 |
| Teams | 10 | 80 |
| International | Yes | No |
| Participants | 8 | 6 |

## 3.3 Data generation

Regarding data generation for this study, multiple different methods were considered, such as observations, documents, and interviews. The use of more than one method could be favorable, to allow for data triangulation. Ultimately, however, interviews were chosen as the sole source of data for this research. Due to the substantial amount of data that was produced by the interviews, as well as the limited time to conduct the research, additional data generation methods were not employed. Interviews were favored as they can be used to obtain detailed information and to explore experiences and emotions that can be hard to extract using other methods. It is also a method that is commonly used in case study research (Oates 2006).

### 3.3.1 Interviews

The data generation method for this study is interviews with participants from the selected cases. The interviews were conducted one-to-one with the interview subjects. These subjects had various roles in the cases, such as developers, testers, designers, system architects, project managers, and team leads. During the planning phase of the study, a number of interviews of around 10-15 was deemed desirable, to get a sufficient number of different perspectives while keeping the amount of data at a manageable level. In the end, the number of interviews was 14. A semi-structured interview approach was used, following pre-defined questions while allowing for new ideas and follow-up questions to emerge during the interview. The interview guide that was used can be found in its entirety in Section A of the appendix. This approach has been preferred over a structured approach, as it may be difficult to know all the good questions before starting the interviews. It also allows the interviewees to speak more freely and in more detail as open-ended questions can be used, rather than closed ones (Oates 2006). As the participants had various roles, the list of interesting questions may vary slightly from interview to interview. Furthermore, this approach is a bit more systematic than an unstructured approach, which may aid the analysis. The interviews were audio recorded after gaining permission from

the interviewee. Audio recordings were used because they give a more accurate reproduction of what was said during the interview than notes and memories can. The audio recordings were transcribed to aid the analysis of the data. The textual data was then analyzed qualitatively.

Each of the interviews lasted between 40 and 50 minutes, for a total of about 630 minutes of accumulated interview time. The interviewees were interviewed once, and the interviews were conducted over a two-week period, from February 28th to March 14th 2023. All the interviews were performed digitally through video calls, both due to geographical separation and because some of the informants were working remotely. A range of various roles were interviewed, to attempt to receive a broad range of perspectives on the topic. An application for collecting personal data in the form of audio recordings was approved by the Norwegian Agency for Shared Services in Education and Research (NSD/Sikt)[9] before the interviews were performed. The handling of the personal data obeyed the guidelines issued by NSD/Sikt, which include solely keeping the data on a password-protected device, and keeping recordings and name lists separate. After the interviews were performed, the audio data was transcribed into anonymous textual data which was used in the analysis phase of the project.

## 3.4 Transcription process

In order to perform a textual analysis of the collected data, the audio recordings had to be transcribed into text. This is a time-consuming process, and according to Oates (2006), researchers should expect to use about 5 hours for every hour of audio recording for transcription. To attempt to speed up this process, an automatic transcription tool was utilized. As some of the interviews were conducted in Norwegian, while others were conducted in English, a multilingual tool was needed. The speech recognition tool Whisper[10], from the artificial intelligence organization OpenAI was chosen, due to impressive speech recognition success rates. The artificial intelligence model is trained on 680,000 hours of audio data, which makes it one of the largest

[9] https://www.sikt.no/

[10] https://openai.com/research/whisper

speech recognition datasets ever created. This has resulted in a stated word error rate of 9.5% for Norwegian audio and 4.2% for English (Radford et al. 2022).

After running the transcription software on the audio recordings, a file with the text transcription along with timestamps was generated. However, as the accuracy of the transcription tool is not at 100%, the transcription files had to be checked and corrected against the raw audio data. After listening through the recordings, the stated word error rates of the tool seemed to be accurate. While some sentences were error-free, others contained up to a few misinterpreted words. Although a fair amount of time was spent listening through the recordings and correcting the transcriptions, the use of the tool was time-saving. As mentioned, at most a few words needed correcting in a sentence, which took less time than manually transcribing the entire sentence. After ensuring that all the textual data was accurate, the analysis phase could commence. The transcription of the 14 interviews resulted in about 80,000 words of textual data to be analyzed.

## 3.5 Qualitative analysis

A qualitative analysis approach was used to analyze the interview data. This form of analysis is better suited for semi-structured interviews than quantitative analysis as the questions may not be identical in all interviews, contrary to structured interviews (Oates 2006). The analysis consisted of attempting to find patterns in the transcribed data, in order to attempt to answer the research questions that have been defined. To aid the analysis, the qualitative data analysis program NVivo was used to code statements into themes. The initial iteration of coding resulted in 391 statements that were coded into 75 categories. These categories were then combined into broader categories in two iterations, which first led to 10 categories, before the final five themes that will be presented in Section 4 were identified. Hierarchical overviews of the initial 75 root-level codes, as well as the final five themes with their sub-codes, can be found in Section B of the appendix. Also, an overview of how the analysis phase was conducted is shown in Figure 13.

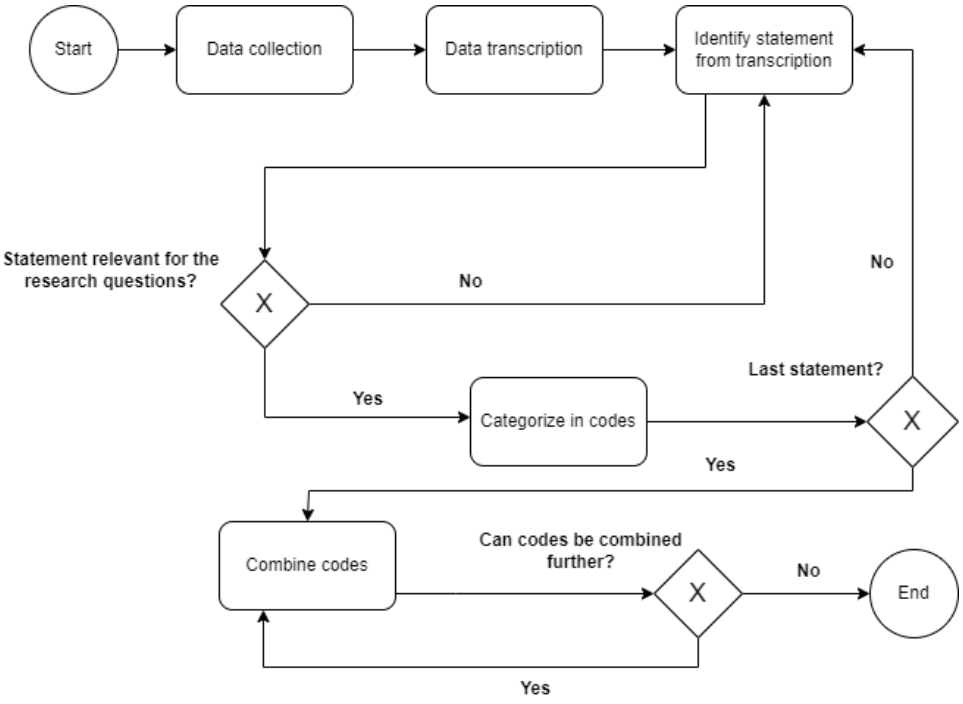



Figure 13: Overview of the analysis phase

## 3.6 Feedback sessions

After the analysis phase was completed, online feedback sessions with each of the two cases were held. These sessions lasted 30 minutes, with a 15-20 minute presentation about the interview findings, and 10-15 minutes of participant feedback. The sessions gave the case participants insight into several aspects of teamwork within their organization, which was perceived as helpful. They were also given the opportunity to give feedback on the findings, for example regarding whether they believed the interpretations from the analysis were accurate. None of the participants expressed that they disagreed with the findings, which improves the confidence in the accuracy of the analysis.

## 3.7 Method evaluation and limitations

The following section will assess the potential threats to the validity of the chosen research method. It will also present relevant limitations of the research.

### 3.7.1 Case study validation

There are four tests that have commonly been used to establish the quality of empirical research (Yin 2018). These are: *Construct validity*, *Internal validity*, *External validity*, and *Reliability*. These four criteria will be used to assess the quality of this research's case study design.

#### Construct validity

Construct validity focuses on identifying the right operational measures for the concepts that are studied. This test is especially challenging in case study research as researchers can tend to use subjective judgments when collecting data (Yin 2018). In this study, the concept studied, namely teamwork in large-scale agile software development, has been defined in Section 2.2. This section draws on relevant studies about teamwork in general, as well as within the agile software development context, explaining what we already know about the concept of teamwork. Additionally, tangible research questions were mapped out in Section 1.2, which are attempted answered based on the collected data.

Yin (2018) lists three tactics to increase the construct validity in case study research, which are to *use multiple sources of evidence*, to *establish a chain of evidence*, and to *have the draft case study report reviewed by key informants*. When it comes to using multiple sources, this research relies solely on interviews as a data generation method, thereby lacking method triangulation (Oates 2006). However, the interviews were conducted with a significant number of participants with a range of different roles, representing different sources in itself. A chain of evidence is maintained as all the statements in Section 4 can be traced back to the original transcription data material. Lastly, a draft of the report has not been reviewed by key informants. However, feedback sessions were held with the participants where it was encouraged to voice any concerns or disagreements with the research findings. A subjective interpretation of the interview data is a genuine risk of this study, as some statements may be ambiguous. However, none of the participants expressed concerns about the findings, reinforcing their validity.

### Internal validity

Internal validity relates to establishing a causal relationship, where certain conditions are believed to lead to other conditions. This is mainly a concern for explanatory studies, such as this one, where the researcher attempts to explain how one event causes another event. Additionally, internal validity also entails inferences, which occur every time an event cannot be directly observed, and the researcher has to infer that the event occurred based on a previous event (Yin 2018). Yin lists the following four tactics to achieve internal validity:

- **Pattern matching** entails comparing an empirical pattern from the study with a previously established predicted pattern (Yin 2018). This is achieved in this study by comparing findings to existing models on teamwork effectiveness.
- **Explanation building** relates to analyzing the case study data by building an explanation about the case (Yin 2018). For multi-case studies such as this one, a general explanation should fit each case involved. Such explanations of the findings relating to the cases are presented for some, but not all the findings.
- **Addressing rival explanations** revolves around testing other plausible explanations for the findings (Yin 2018). Rival explanations are rarely addressed in this research.
- **Using logic models** relates to developing a model of a complex chain of occurrences or events over time (Yin 2018). Such a model has not been utilized in this study.

### External validity

External validity deals with whether the study's findings are generalizable to other cases (Yin 2018). To increase external validity, Yin (2018) suggests the use of replication logic in multi-case studies. This study has followed a literal replication approach, where two cases have been studied using the same research design, methodology, procedures, and data analysis techniques. A threat to the generalizability

of the findings of this research is the limited sample size. While the number of participants may be sufficient to make general statements about a case, it is not enough to be a representative sample size for the entire field of large-scale agile software development.

**Reliability**

Reliability relates to the reproducibility of the study, meaning that if another researcher followed the same procedure for the same case, they should arrive at the same conclusion (Yin 2018). To achieve reliability Yin (2018) suggests the employment of two strategies, using a *case study protocol* and a *case study database*. Section 3 of this study documents the steps that were taken during this research, from research design to data analysis. A separate case study database has not been created for this study, however.

### 3.7.2 Limitations

One limitation of this research is that the data was collected over a relatively short period of time, with all the interviews being carried out over a two-week period. This prevents the study from investigating longitudinal effects, which would require following a case for a longer period of time, from one month up to several years (Oates 2006). Additionally, some limitations follow from the choice of data generation method. While the interviews were of a reasonable length, around 45 minutes each, this is still a limited time frame compared to other data generation methods, like for example observations. The lack of time imposed by doing interviews can lead to issues with interviewees creating opinions due to the time pressure, which may not have been as strong in reality as they may be perceived (Myers and Newman 2007).

Myers and Newman (2007) also lists other relevant limitations of interviews, such as the *artificiality of interviews* and *lack of trust*. The setting of interviews is artificial by nature, which may impact the accuracy of statements from subjects. Also, all the interview subjects were complete strangers to the researcher prior to the interviews. A lack of trust can cause subjects to withhold potentially important information

that they may perceive as sensitive. The lack of trust may also be reinforced by the fact that all the interviews were held online via video calls, which could impact the connection and relationship between the researcher and the informants.

# 4 Results

This section will present the findings from the conducted interviews from Signicat and NAV IT, which have been introduced in Section 3.2.1 and Section 3.2.2. Findings from the two cases will be presented together so that similarities and differences can be highlighted. An overview of the conducted interviews and the roles of the participants for the two cases can be found in Table 8. The findings will be presented categorically, after themes that impact effective teamwork identified during the analysis phase.

Table 8: Overview of interviews

| **Signicat** | | | **NAV IT** | | |
|---|---|---|---|---|---|
| **ID** | **Date** | **Role** | **ID** | **Date** | **Role** |
| P1 | 28.02.2023 | Developer | P9 | 08.03.2023 | Developer/tech lead |
| P2 | 28.02.2023 | Product owner | P10 | 08.03.2023 | Product lead |
| P3 | 28.02.2023 | Product owner | P11 | 09.03.2023 | Product lead |
| P4 | 01.03.2023 | Developer | P12 | 09.03.2023 | Solutions architect |
| P5 | 01.03.2023 | Product owner | P13 | 10.03.2023 | Developer |
| P6 | 01.03.2023 | Architect | P14 | 14.03.2023 | Team lead |
| P7 | 02.03.2023 | Developer | | | |
| P8 | 02.03.2023 | Developer | | | |

## 4.1 Agile methods and work process

There are a lot of ways to incorporate an agile philosophy into a development team, but finding the right way to do so may not be trivial. Further, continuous improvements may be needed to reach the most optimal work process.

### 4.1.1 Choice of agile method

In agile development teams, the choice of agile method and work process may be an important one to achieve a high level of performance. From this research, the most commonly used agile methods by the interviewed teams were Scrum, Kanban, and a combination of these. Some teams also employed practices from XP. On the organizational level, Signicat also used a structure that is inspired by the Spotify model. The popularity of the various agile methods is shown in Figure 14, where the number of participants that stated that their team used a specific method is represented on the y-axis. The developers of teams that were using Scrum were generally satisfied with their process and thought Scrum worked well (P7, P8). One of the participants highlighted that for their team, the appointment of a Scrum Master who took their role seriously greatly improved the success of their process (P8). Others give credit to agile coaches, who have joined teams sporadically to help improve their agile process (P9, P12).

A developer and a product owner from another team shared that they had transitioned from using Scrum to Scrumban, which was very beneficial for their team (P1, P3). They felt that working in sprints was limiting and that it was hard to define meaningful goals and tasks for a two or three-week period at a time. By leaning more towards Kanban they could work in a more continuous rather than iteration-based way. By not committing to a defined sprint, they avoided issues with waiting for the next sprint to start on tasks that were not in the current sprint backlog. Additionally, one of the developers from one of the Scrum teams felt that while Scrum was working well, they could benefit from a more "open-ended process", such as Kanban (P7). As such, it seemed like participants, in general, were more satisfied with less prescriptive processes, due to increased flexibility.

While not showing as widespread use as Scrum and Kanban, some participants also reported the use of XP techniques such as pair programming and even mob programming with even more developers involved (P10, P13). These techniques were deemed highly successful both for productivity and for sharing knowledge between more experienced and less experienced developers. This was seen as a risk-reducing

measure, as it would lower the impact of experienced developers being unavailable. Employing the right agile methods for the team seems to be an important measure to improve how well team members collaborate. However, many teams used practices from some agile methods without following the entire process. Examples of these are tribes and guilds from the Spotify model, and stand-ups and retrospective meetings from Scrum. There is no "one method fits all", but letting the team tailor their own process seems to be a good choice according to the participants. With a well-fitting process, planning, coordination, and consequently teamwork becomes easier and more effective, which in turn can lead to a good pace of development.

> *"I feel that organizations are very rigid in terms of adopting agile when they shouldn't. (...) people try to fit, they try to make it a cookie-cutter thing. One method applies to all, and that doesn't work. Different teams, different people, you have to be willing to drop certain things from the agile method to make things better for your team, to increase the happiness, efficiency, or just get [rid of] things that your team isn't using. (...) But, yeah, teams have to adapt to agile. They have to want to do it and think it's worth it. Otherwise, you're just going through the motions and you're not getting much out of it." (P4, Developer, Signicat)*

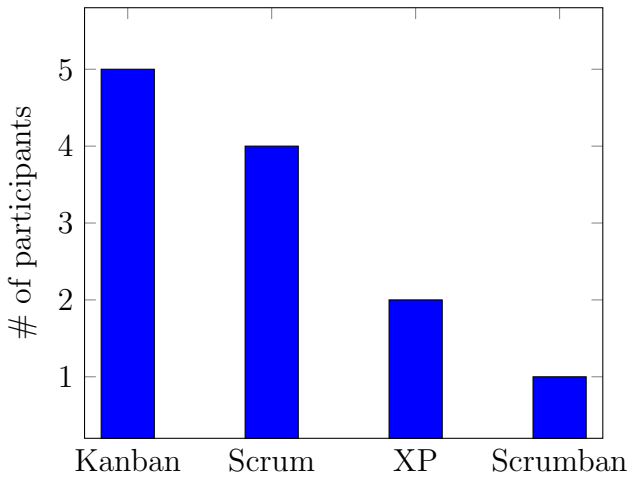



Figure 14: Frequency of use of agile methods among participants

### 4.1.2 Work tasks and specialization

Another aspect of the work process that impacts teamwork is how tasks are defined and assigned to team members. One team struggled with task dependencies, where one task needed to be finished before another could be started (P7, P8). This meant that tasks could not be worked on in parallel, which could reduce the velocity of the team. This was hard to avoid when defining tasks, partly due to the type of software that was being developed where many components were dependent on each other. To combat this issue, tasks containing internal dependencies were not planned for the same sprint, ensuring that the tasks were taken on in the correct order.

Regarding tasks, having multiple developers working on the same feature was seen as a decision that improved the success of teamwork (P2, P8). To enable this, thorough specification of tasks was needed, so that multiple people could pick up on related tasks, rather than only one person immersing themselves in the problem. This reduced the risk of progress slowing down due to one person becoming unavailable. Also, some issues could emerge if only one person worked on a feature for a while based on incorrect assumptions they made early on, which could be prevented with more input from others. Additionally, one of the product owners noted working with someone was both more enjoyable and more effective, compared to working alone (P2).

Some of the participants also noted that due to a large domain, teams and developers were unable to specialize in one part of the domain, and were forced to work on a wide range of parts of the domain and pulling in different directions at the same time (P10, P11). This increased the amount of context-switching which was required and reduced the velocity of the teams. One mitigation to this issue was an experimental establishment of temporary "task force" groups who only concerned themselves with one specific part of the application, and was isolated from the rest of the team. This was deemed a great success and the group achieved a great pace as they could focus on a single mission. This experiment later influenced a larger restructuring of the team, where they made sub-groups within the team that had the main responsibility for a single part of the application.

### 4.1.3 Improvement

The use of agile coaches has already been mentioned as one method of improving the work processes of agile development teams. Another way of improving team performance is making use of reflection. One common method of achieving reflection is the use of retrospective meetings, which all the interviewed teams shared that they used, at either regular or irregular intervals. The majority of the participants expressed that these meetings were helpful in improving the way the team worked. While some teams used the retrospective meetings to identify specific problems and measures to focus on for the upcoming sprint, other teams used the arena as a status report where team members could share what they had done during the last sprint and their perceptions of the work process. Both those who had concrete outcomes of the meeting and those who used it in a more abstract way felt the meeting was helpful and that the time spent on the retrospectives was well spent. One team lead noted that it was important for them to appoint a person who had the responsibility of following up on the issues that are chosen for improvement (P14).

Another way of improving teamwork, as well as the collective knowledge level of the team, is the use of knowledge-sharing techniques and practices. For multiple teams, having several developers working on the same or similar tasks was important to share knowledge within the team. This forced the developers to cooperate with each other, where they could learn from each other. Typically, less experienced developers could learn from more experienced ones when working on the same task, but it was also noted that the more senior developers learned from less experienced teammates, who could bring knowledge about more modern technologies into the teams. Other measures that have been used to share knowledge include impromptu "show and tell" meetings, inter-team communities, guilds, and demo meetings. While some of the participants were happy with the amount of knowledge sharing that took place, others expressed that their teams could benefit from being more open and willing to initiate knowledge sharing with others (P1, P11). Knowledge sharing within the team, as well as within the organizations is a good way of enabling team members to learn, in addition to encouraging teamwork and collaboration with others.

## 4.2 Autonomy, alignment, and leadership

The agile manifesto suggests that teams can achieve the best results if they are self-organized. In general, team autonomy is seen as an important part of agile, which contrasts more traditional methods where higher management takes most of the decisions, while developers simply develop what they are told. Being able to influence decisions was something that most of the informants saw as an important aspect of their work process, but there were some differences in how the teams were organized and led. One of the issues that require decision-making is the prioritization of tasks. Many of the interviewees stated that their teams had product owners who were responsible for deciding the order in which to complete tasks (P1, P3, P4, P5, P7, P8, P12). Generally speaking for these teams, there was dialogue between the developers and product owners on the topic of prioritization, with the product owner being the tiebreaker when disagreements occur. One advantage of this is that the product owners are able to get a better overview of not only their own teams' needs but also other teams' needs as well as company-wide goals, than what the developers are able to. One developer also expressed that the developers were happy to be relieved of duties regarding planning and other organizational tasks so that they could spend more of their time developing (P1).

When it comes to how to perform tasks, however, the developers were free to decide themselves. In general, developers were able to reach agreements on technical matters amongst themselves. In the event that developers could not agree on how to implement a feature, the tech lead of the team would be involved and act as a tiebreaker. Most of those that shared that product owners were responsible for task prioritization came from Signicat. In NAV IT, however, much of their work consists of adhering to new regulations put in place by the Norwegian government. Hence, a lot of the prioritization comes indirectly from external sources.

Participants from NAV IT also highlighted that while they had little influence on things that needed to be done due to government regulations, the teams had full ownership over the rest capacity. This is the time that can be used on tasks that the team decides themselves, while still being able to complete the tasks that are

imposed upon them from above. One of the most common uses of this rest capacity from the interviewed teams was upgrading systems to more modern technologies. Both cases were in the process of modernizing their systems, to improve development pace, and robustness, and to make recruitment easier.

The leadership of the teams did not only rely on specific roles that were present but also on the composition of people in terms of experience. One product owner from Signicat was a part of a team that was full of experience, and which consisted of more senior than junior developers (P2). This high level of experience impacted the team in several ways. Other teams often had requests and questions for the developers of this team, as some of them had been at the company since its beginning. One measure that was implemented was that these kinds of requests needed to go through the product owner first, who could shield the developers from being overloaded with questions from others. Additionally, leading a team of highly experienced developers may not always be simple, as they may have more personal opinions on how to do things and what to do than less experienced developers.

> *"We try, for the [team members'] satisfaction, to facilitate that people can work with things they are interested in. And also because, leading senior developers is like herding cats, they do as they want sometimes. But yeah, it's always best when people work with the things they want." (P2, Product owner, Signicat)*

A developer from Signicat also highlighted senior developers' impact on team leadership (P7). They believed that the seniors could be a bit too dominant when it comes to making decisions for the team. Even though the experienced developers may be best suited to lead the team in the right direction, it was expressed that it would be beneficial to the more junior team members if the seniors sometimes took a step back and allowed others space and responsibility.

> *"I think the team could benefit if the seniors let go a little more than what they sometimes do. Because I think I have been a junior myself and been given that trust from seniors, that enabled me to blossom and*

*develop into the senior developer I am today (...) With freedom under responsibility, I think most developers would blossom faster, than if you only treat them as code monkeys." (P7, Developer, Signicat)*

Team autonomy and self-organization can be tied to several benefits to an agile team. No two teams are alike, so allowing teams to customize agile methods to suit their team may be favorable. Team members may also be more motivated if they feel that they have a voice in decision-making. However, there may also be drawbacks to giving individual teams this type of control. Without external guidance, team members within a team may become misaligned with each other. Without a firm common goal, individuals may pull in different directions, either purposely to follow their own desires, or without being aware that it is happening. This means that the team makes several small contributions to various areas, instead of making a more significant impact on one prioritized area.

One participant saw this intra-team misalignment as a significant challenge, as various team members had different perceptions of what should be prioritized and which goals to aim for (P12). Another participant saw the same challenge occur on a higher organizational level, where the organization as a whole lacked a common direction in some areas (P9). When teams are misaligned with each other, it can make the integration of different sub-systems more difficult, and if teams are pulling in different directions they may even end up working against each other.

Another potential challenge with bottom-up team organization is that it can be harder for higher management levels to monitor and manage the different teams. This challenge was mentioned by one product owner and one architect (P5, P6). While some teams may use completed story points as a measure of progress, other teams may use another metric or have no metric at all for measuring progress. This can make it difficult for middle management to determine the velocity of teams, and to make decisions on whether changes are needed or not. Differences in work process can also complicate inter-team interactions, as teams may not be in the same phase in the project or sprint life-cycle. There are both advantages and disadvantages associated with team autonomy, so organizations should adjust the level of control

given to teams to maximize upside and limit downside.

## 4.3 Inter-team coordination

One of the main differences between small-scale and large-scale agile development is the increased need for coordination across several different teams. Almost all of the participants expressed that inter-team coordination was more difficult than intra-team coordination, with some reporting it as one of the biggest challenges in their organization. A range of various reasons to why inter-team coordination is difficult have been given by the interviewees. Some think that the issues come from not knowing and trusting people from other teams as much as those in your team (P7, P9). One developer blames inter-team coordination issues on method misalignment (P8). As the different teams have the autonomy to choose their own agile methods, there is no unified work process across teams. If one team is dependent on another team completing a task, they may have to wait until the next sprint for the task to be prioritized into the sprint backlog. As the teams don't have the same sprint lengths, and not everyone works iteration-based, adjusting to other teams' processes may be challenging.

Two developers from the same team noted that collaborating with other teams is harder because some of the teams are located in other countries (P7, P8). This makes communication more formal, as meetings need to be planned rather than contacting people who work in the same office in person. A product owner from Signicat expressed issues with inter-team communication going through various channels (P5). They shared that requests can come from tribe leads, tech leads, and other teams' product owners, which can lead to the same question being asked and answered up to three times. Some informants highlight that requests from other teams need to be filtered by the product owner to avoid fragmentation of the developers' work days (P2, P3). This is a bigger challenge for senior-heavy teams that possess more knowledge than others, as these are often the ones who are asked for help. Teams should therefore strive to find an appropriate balance between helping other teams and progressing with their own tasks.

> *"Yes, there is a significant difference in collaboration with the team, compared to working with another team. (...) It's hard to ask those that you don't know as well, and maybe don't trust as much as those you work with daily and know very well. With whom you have a familiar dynamic. That barrier is definitely present, and I think it's something we should work on. That is perhaps the biggest problem we have holding us back..." (P7, Developer, Signicat)*

While some share that their team regularly collaborates with other teams (P8, P9, P12), others experience that such interactions are rather rare (P1, P6). When inter-team dependencies are rare, there is also less to be gained from effectivizing these interactions. Several informants share that digital tools such as Slack are helpful when coordinating with other teams (P3, P10, P13). Additionally, inter-team product owner meetings and inter-team forums are used to share information between teams. Knowledge of other teams' situations is seen as helpful for being able to solve dependencies between teams. One developer also notes that going through a middle management level, rather than contacting other teams directly is more effective (P4). In the case of Signicat, tribe leads were seen as useful for such requests, as they have a better overview of the status of the various teams. When companies grow larger, more development teams get formed making coordinating work more complex. Being observant of issues and aiming to continuously improve communication between teams may be important to avoid teams being blocked by other teams and to resolve dependencies as effectively as possible.

## 4.4 Intra-team collaboration

While coordinating with other teams may be an important area of focus in large-scale agile development, being able to collaborate well within the team can be equally important.

### 4.4.1 Adaptability

Adaptability is a central aspect of agile software development. By limiting the amount of detailed, long-term planning, agile teams are set up to respond to changes well and to be able to change directions with limited downside. This is consistent with the responses from the informants, with most of them expressing that their team adapts quickly to changes (P3, P4, P5, P6, P7, P8, P11, P12, P14). Some credit the high level of adaptability to their agile process, with Scrum and Kanban being highlighted as beneficial for versatility. A team lead from NAV IT also believed that expectation management in addition to having a good work environment and sense of community helps (P14). Daily releases were also emphasized as improving adaptability, as it makes it easier to integrate changes from original plans (P10). Moreover, one developer highlighted that having the option to re-organize and re-assign tasks easily was important for team performance (P1). In order to facilitate this, adaptability is needed.

Being versatile and dealing with changes is not always a painless process, however. An architect from Signicat shared that versatility comes at a price, as the team can end up working with a lot of different things at the same time (P6). This leads to a lot of context-switching and reduced pace, as context-switching takes time. Further, it causes the team to make small contributions in several directions, rather than a more significant contribution towards one goal. One developer also expressed that while some changes might be easy to manage, changes to personnel might be harder to deal with (P7). When key team members leave the team, they may leave behind code that no one else in the team has knowledge of.

### 4.4.2 Feedback

Feedback can be an important tool for improving the work process of a team. Without it, it can be difficult to know which areas to improve and what challenges the team is facing. However, some might not be as comfortable giving feedback as others, so maintaining a culture where it is safe to do so may be key. Most of the respondents expressed that feedback on technical matters occurred more often than

other types of feedback and that there was a very low threshold for bringing up this type of feedback. Feedback on personal matters, however, has been described as being a "touchy" topic and is therefore not brought up as often (P2). The most commonly used arena for technical feedback was code reviews. One developer from Signicat shared that code reviews were frequently used to give feedback on the code of other people, and that both junior and senior developers were happy to share their opinions with others (P7).

Another arena that was commonly used for giving feedback to others was retrospective meetings. In these meetings, both technical matters as well as feedback on the work process were brought up. Several of the interviewees highlighted that feedback is very important for their team (P10, P11, P14). If issues are not brought up and discussed, problems can grow and persist, rather than be dealt with. Some of the informants have also expressed that their team could be better at giving feedback to each other, both positive and negative. Some might be reluctant to give feedback to their peers fearing that it might be perceived as critique. Hence, creating a work environment where feedback is common and where it is safe and encouraged to share opinions may be beneficial for improving team performance.

### 4.4.3 Team spirit and trust

A unified team that is motivated to work together may be important to improve the chances of effective teamwork. Additionally, trusting your peers could also be an important aspect to enable successful collaboration. Overall, the interviewees expressed that their team had good team spirit. Five informants rated their teams' team spirits as "very good", while five stated that the team spirit was "good". Some characteristics the informants have used to describe a high level of team spirit include being willing to help others, rooting for each other, and that people are more concerned with things being done than that things are done in their way.

However, one of the architects noted that while team spirit within the teams was at a good level, it could be better across teams (P6). They felt that employees could benefit from bonding more with people outside their teams, as they don't

know others as well as they know their teammates. This could be connected to the findings regarding inter-team coordination, where it was found that cooperating across teams was significantly harder than within teams. As NAV IT has some products that use old technology, a product lead shared that outdated tech can be an aspect that lowers team spirit and causes team members to move on from the team (P10). They also shared that budget cuts have a negative impact, while social happenings and gatherings improve team spirit. A developer from Signicat also stated that the unity between the team and upper management was not as good as within the team and that the motivation of team members can be impacted by poor decisions from management positions (P8). Several of the interviewees highlight that team spirit is important for both team performance and employee satisfaction, with several positive consequences listed.

> *"Do you believe that it [good team spirit] contributes to people working better?" (Interviewer)*
>
> *"Definitely. Working better, and I think it might even lead to less sick leave. It brings a lot of positive things. People work better, are more satisfied at work, and I think these things can give better health, which in turn increases performance. You get a positive loop." (P13, Developer, NAV IT)*

Another aspect that could impact teamwork is the level of trust that team members have in each other. Most of the interviewees that rated the level of team spirit in their team high, also rated the level of trust within the team high, suggesting that these two aspects are linked. Several of those who believed their team members were trusting of each other, stated that this trust had been built up over time. Keeping the team composition stable and avoiding too many changes to personnel have been highlighted as a reason for increased trust (P7). It was also claimed that having a common identity as a team and feeling a sense of community made all tasks easier to accomplish (P14). Reaching good levels of team spirit and trust seems to be something that takes time to build, but creating a good work environment appears to be a meaningful investment to increase productivity and satisfaction.

### 4.4.4 Shared mental models

Shared mental models refer to "being on the same page" as your peers. This includes tacit knowledge that all team members have and that each individual can expect their teammates to possess. One example of a shared mental model is having a common language or jargon, where the members of the teams use the same names for components, systems, practices, and so on, so that everyone understands each other (P3, P6). This type of knowledge is typically not written down or documented anywhere, as it is implied that those who need to know this information already know it. These shared mental models can be positive and even necessary for team performance, especially when teams deal with complex systems. However, too much tacit knowledge may be challenging for new members of a team or those outside of the team, who may have trouble collaborating due to a lack of knowledge and documentation.

From the interviews, three informants stated that their team had a lot of shared mental models (P1, P5, P13). Seven of the interviewees shared that their team had some shared mental models and tacit knowledge (P2, P3, P4, P6, P8, P11, P14). There was no clear difference between the two cases in the extent to which shared mental models were present. None of those interviewed expressed that they did not have shared mental models, suggesting that this type of knowledge is common in agile software development. Four informants expressed that their teams could benefit from more extensive documentation or from a more structured system for writing documentation (P2, P6, P13, P14). One example of this is that one product owner stated that only those within the team were able to understand their task descriptions in Jira (P2).

On the contrary, two participants felt that their teams produced sufficient documentation (P8, P11). For those who have sufficient documentation, however, a concern was raised regarding the complexity of finding what you are looking for in these documents. One product lead shared that their architects had produced over 3000 pages of documentation, which was meaningless to use for others as it was simply too long (P11). Others also noted that while they could benefit from more extensive

documentation in some situations, writing these documents can be time-consuming and the time of the team members could hence be put to better use elsewhere. Information in documentation also gets outdated quickly, as components, teams and the organization may change frequently.

### 4.4.5 Competence redundancy

Some agile methods suggest that teams should be cross-functional. Instead of solely relying on specialists who only have a specific competence, team members are encouraged to have overlapping skill sets, to improve flexibility and reduce reliance on individuals. Most of the participants expressed that they had a redundancy of competence to some degree, but no one felt that their team was entirely cross-functional. Several respondents stated that over time as the team matured, knowledge sharing between team members increased the overlap of skill sets, reducing risks. Some of the interviewees stated that they tried to achieve that at least two team members could do any one task, or even that most of the team members could do the majority of tasks (P2, P6). However, they also expressed that this is hard to accomplish in practice.

Amongst other reasons, one cause for this was that some of the team members had worked with the system substantially longer than others, meaning that they had a deep understanding of how the system worked, which was difficult to transfer. Another reason was a lack of resources. In order to achieve a satisfactory degree of skill set overlap, teams would generally need to either hire more employees than what they strictly speaking would need to complete their tasks or allocate time to sharing the knowledge that is present between team members. Both of these measures require resources, in the form of time or money, which is dependent on allocations from higher management.

The knowledge-sharing technique that was most often mentioned by the respondents was pair programming. Several participants believed this was an effective way of spreading competence. One team lead, however, noted that pair-programming sessions could be intense and demanding and that it was a balancing act to not overdo

it and exhaust the developers (P14). A developer from NAV IT also highlighted the importance of being able to connect those who have and those who lack knowledge in a specific area together, rather than those who have similar competence (P13). Additionally, considering what areas people are interested in learning more about when pairing up developers can be beneficial for motivation. Assigning several developers to the same task or the same set of tasks has also been mentioned as a measure that increases redundancy (P2, P8). Increasing the production of documentation has also been mentioned as a measure that increases redundancy, as it enables those who lack the knowledge to work on a task without requiring the help of others (P2, P7).

Although competence redundancy can increase adaptability, one concern was also highlighted. A developer from NAV IT shared that when everyone can do anything, no one takes ownership of any area (P13). Without individuals that have designated responsibility for an area, it can be harder to know who to ask when facing issues, which can be a cause of friction and annoyance. Overall, however, the participants see the benefit of competence redundancy within the team and believe that it increases adaptability and reduces the negative impact of unavailable individuals.

## 4.5 Remote and hybrid solutions

With the use of various digital tools, software developers have been able to collaborate with their colleagues without being co-located for some time. However, during the Covid-19 pandemic, remote work became essential as many regions required workers to work from home if possible. Even as restrictions have been lifted, some companies have continued to support remote work as a viable option for coming in to the office. All of the interviewed participants shared that their team supported remote solutions and that at least some of the team members worked from home either permanently or occasionally. From the interviews, three informants felt that allowing team members to work remotely worked very well (P11, P12, P14). Some even stated that they would not be open to working anywhere that didn't support this type of freedom in the future. Further, three interviewees stated that their

hybrid solution worked well (P3, P4, P10), three had a mixed experience (P2, P8, P13), and two felt that hybrid working was challenging (P5, P6). The aspect that was most commonly highlighted as an advantage of being able to work remotely is that it can help avoid distractions and improve focus. The interviewed teams have open office landscapes where multiple employees sit in close proximity to each other, which may increase distractions. Other advantages that were mentioned include gaining time from not having to commute and being able to be available for family members during the day, for example in the case of sick kids. However, some challenges with teams being divided between remote and in-office working have also been brought up.

Multiple informants have expressed that those who work from home can become isolated from those who work from the office and that they can miss out on useful discussions that emerge in the workplace (P7, P8, P14). These issues have in some cases been attempted solved by sharing outcomes of discussions on digital channels, or inviting remote workers to online meetings when these discussions occur. However, the interviewees have shared that it is easy to forget to involve those who are not present, and documenting every little discussion is time-consuming. One developer also expressed that hybrid solutions can be challenging for those in the office, as they have to find available meeting rooms when participating in online meetings, to avoid disturbing others (P13).

Some of the interviewees were in a position where they had to collaborate a lot with team members situated in other countries who only were available online (P5, P6). These expressed that digital communication was inferior to face-to-face communication and that contacting others would be easier if everyone sat in the same office. As online meetings had to be set up in order to meet up with colleagues, these meetings had to fit into everyone's calendar, and such meetings often had to be set up a week in advance to ensure that everyone could participate.

> *"Via screen it's difficult because you don't see everything that's going on there and sometimes it's easy just to scream over your shoulder "What do you think about that?" and then it's just a short answer. If you have*

*to select someone and then, not all details... you forget something or you just don't think to mention it and because it's not that important you don't want to type it down, so yeah you lose a lot of, I think of the details in a conversation when you're not face to face with someone, but it is what it is." (P5, Product owner, Signicat)*

In Figure 15 an overview of the level of satisfaction with working remotely or hybrid from some of the participants of each case is displayed. The level of satisfaction is rated on a scale from 1 to 5 based on participant statements. Each rating level can be described as follows: *1 - Does not work well, 2 - Challenging, 3 - Mixed, 4 - Works well, 5 - Works very well.* From the roles, it was difficult to see a definite pattern as to which roles were satisfied with this type of work, and which were dissatisfied. However, it is apparent that participants from NAV IT in general were more satisfied with remote and hybrid working than those from Signicat. The average satisfaction level of employees from Signicat was 3.0, while it was 4.6 for NAV IT. There might be multiple explanations for this contrast, but the main difference that was found between these two cases regarding remote work and digital collaboration was that participants from Signicat often have to collaborate with others that are located in other countries and offices, while all the participants from NAV IT belong to the same office. This suggests that physical location is not insignificant even when collaborating digitally. Being able to physically gather from time to time with those you work with has been highlighted as a measure that improves teamwork in hybrid teams. This can be either social gatherings, meetings, or seminars. Such gatherings may be more difficult to realize and occur less often when people are situated in different countries and offices.

*"As soon as we became a new team we managed to meet physically at least once, and the fact that they [developers from another country] came to us and were here a couple of days, that was, from a digital context it is easier when you have met someone in person, chit-chatted a bit and kind of seen their body language to see who people really are. Picking up on how people are in writing compared to how they really are is not always easy. After being with them for a couple of days you can know*

*that if someone writes in a way that makes them sound annoyed, they may not really be [annoyed], just a little cautious." (P2, Product owner, Signicat)*

Ultimately, the feedback on the use of remote and hybrid solutions has been mixed, with a skew toward the positive side. It seems to have a significant impact on teamwork effectiveness, and finding a solution that suits the team may be important to enable a high level of productivity and employee satisfaction. Measures that are found to improve satisfaction with remote solutions include making sure that the team also gathers in person from time to time, and allowing for freedom of choice in terms of when and how often to work remotely.

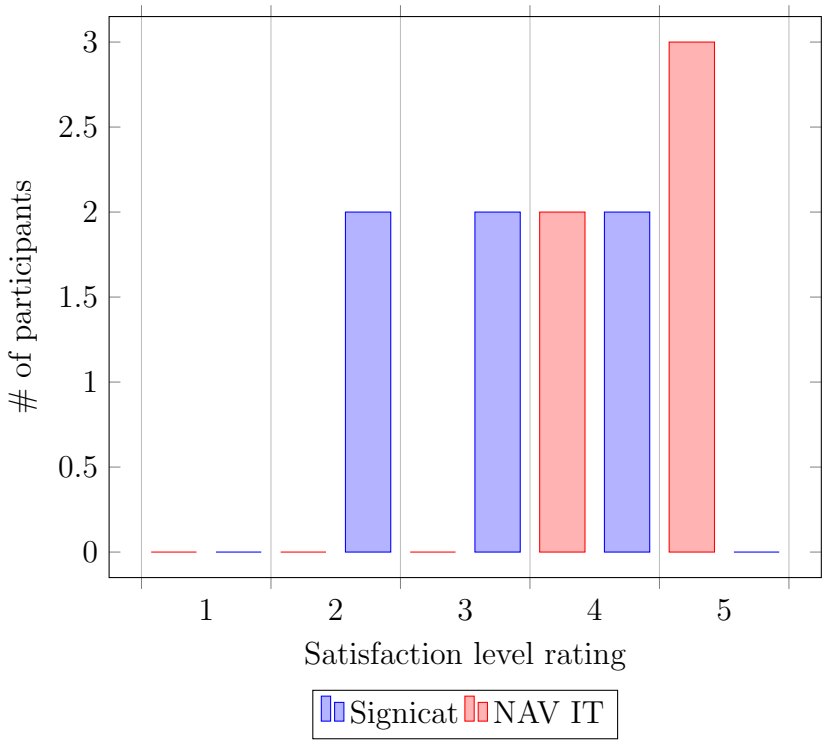



Figure 15: Satisfaction with remote or hybrid on a scale from 1 to 5, based on participant statements

# 5 Discussion

This section will use the findings from Section 4 to attempt to answer the three main research questions from Section 1.2. This will be done in three parts. The first part will identify factors that can impact teamwork effectiveness in large-scale agile teams. These factors will be split up into the factors that foster teamwork effectiveness, and those that hinder effective teamwork. The second part will present strategies that have successfully been employed in the two cases to enable teamwork effectiveness. These strategies can potentially be used by other large-scale agile organizations to set themselves up for teamwork success. The third and final part compares the findings of this study with an existing model on teamwork effectiveness in agile development teams, to investigate whether this model is also applicable to the large-scale context.

## 5.1 Impact on teamwork effectiveness

One of the aims of this study was to investigate what impacts teamwork effectiveness in large-scale agile software development teams. In order to be able to improve the success of teamwork, knowing which aspects actually have an impact may be an important first step. This section will present both the aspects that have been found to improve and those that have been found to decrease teamwork effectiveness throughout this research. It will thus attempt to answer the first research question, which was given as follows:

> **RQ1:** What factors do participants perceive as impacting teamwork effectiveness in the large-scale agile context?

### 5.1.1 What fosters effective teamwork?

The first part of the first research question of this thesis was given as follows:

> **RQ1.1:** What fosters effective teamwork in a large-scale agile context?

The most commonly mentioned factors that foster effective teamwork are shown in Table 9, sorted by prevalence.

Table 9: Factors that fosters effective teamwork

| **Factor** | **Participants** |
|---|---|
| Right agile methods | P1, P3, P7, P8, P10, P13 |
| Knowledge sharing techniques | P10, P11, P13, P14 |
| Scrum Master/agile coach | P8, P9, P12 |
| Digital tools | P3, P10, P13 |
| Feedback | P10, P11, P14 |
| Remote work | P11, P12, P14 |
| Working on the same task | P2, P8 |

**Right agile methods**

The most prevalent factor that was found to contribute to increased teamwork effectiveness was using the agile methods that are right for the team. The most commonly used methods in the teams that were interviewed were Scrum and Kanban. Most of the participants were satisfied with these methods, but some indicated that Scrum was too rigid. While Scrum is a lot less prescriptive than plan-based models, such as the waterfall model, other methods like Kanban have fewer constricting rules. Another important aspect of finding the right agile work process for your team is adjusting the methods rather than simply selecting a preexisting method. Several of the teams used parts of one or multiple agile methods, without following any one method in every detail. This was seen as beneficial, as the teams were able to discover a work process that was better suited for themselves. It has also been noted that improving work processes takes time, so teams should not expect to find the right process for them straight away. It has also been highlighted that keeping the team stable over an extended period of time helps with improving the process.

**Knowledge sharing techniques**

The use of certain knowledge-sharing techniques was found to contribute to how

well team members could collaborate with each other. Hansen et al. (1999) state that two main strategies can be employed to manage the knowledge in a company, *codification* and *personalization*. From the interviews, knowledge-sharing techniques belonging to the strategy of personalization were most commonly mentioned. Such techniques include pair programming, mob programming, communities of practice, and demo meetings. Further, knowledge management tools belonging to the strategy of codification, such as wikis and documentation, were also mentioned.

While the main effect of knowledge sharing may be an increased level of competence in the team, a side effect seems to be more effective teamwork. Some knowledge-sharing techniques involve working together, which implicitly forces team members to collaborate. This can be useful for those teams where such collaboration may not happen organically. Additionally, by raising the competence level of the less experienced developers, more of the developers are able to accomplish tasks without assistance from their seniors, leading to less friction and higher productivity for all parties. Pair programming was one technique that was labeled as beneficial to their teams by several of the interviewees. These sessions sparked discussions that may not have been brought up otherwise and helped align the team when it comes to technical decisions.

#### Scrum Master/agile coach

Some of the participants gave credit to a Scrum Master or agile coach for improving intra-team collaboration. The Scrum Master role is typically fulfilled by a member of the team. The role can either be held by the same team member continuously or be rotated at regular intervals. Some teams saw a noticeable benefit when they either appointed their first Scrum Master or when the role was given to someone who took the role seriously. A Scrum Master can help the team follow its process, which can include making sure that retrospective meetings are carried out and facilitating meetings. Ensuring that the process is followed can increase the effectiveness of the agile methods that are used. While some team members may be aware of the benefits of adhering to their work process, people may not feel the urge to initiate changes unless it is their explicit responsibility. The agile coach is another type of

role that was occasionally used by teams from NAV IT. While these coaches also contribute to improving the agile process, they are not permanently employed in the team and are rather called upon when a need is identified or at regular intervals. An advantage of using agile coaches can be that they bring experience from other teams, and can advise an agile team based on what has worked in other teams.

**Digital tools**

The use of digital tools for collaboration was also a factor that was mentioned to improve teamwork. The tool that was most often referred to as a tool that helped communication was Slack, while other tools include Teams, Zoom, Skype, Discord, and HipChat. The use of various arenas for communication has also previously been found to aid coordination in large-scale agile development (Edison et al. 2022). One developer stated that using Slack was occasionally more productive than meetings (P4). Slack was also labeled as a "fantastic tool for collaboration", as well as "by far the most popular tool" (P9, P14). In general, the digital tools that the teams employ help team members get in contact with anyone in their organization. They also greatly reduce response times compared to physical meetings that may need to be planned ahead of time. Digital tools are also vital for those who work from home, or when employees from different offices have to work together.

**Feedback**

Several of the participants highlighted that feedback was vital in order to be able to improve as a team. The most commonly used arenas for giving and receiving feedback were code reviews for feedback on code, and retrospective meetings for all types of feedback. These types of reflection are examples of reflection-on-action, as they review what has happened (Babb et al. 2014). As the level of autonomy generally was high, teams were able to adjust their own work process. To enable adjustment and improvement of the process, feedback regarding methods was occasionally brought up during retrospectives. While code reviews and retrospectives were effective for sharing feedback, some interviewees expressed that team members could improve on giving feedback outside of the established arenas. This could indicate that teams could benefit from creating a work environment where feedback is

encouraged, and where team members do not have to fear that constructive feedback is seen as critique.

#### Remote work

With the digital collaboration tools that exist in today's world, it is possible to work in agile development teams from anywhere. This type of freedom was a crucial factor for some of the participants with two of them stating that they could not see themselves working for a company that did not allow employees to work remotely (P12, P13). Working from home has been claimed to increase focus for some, as there are fewer distractions than at the office. It has also been stated that collaboration with pair programming works better in a remote setting, as everyone involved can have access to their own keyboard while sharing their screen.

#### Working on the same task

One simple measure that was seen as positive for improving teamwork was assigning multiple team members to the same task, or the same set of tasks. This simple strategy ensures that at least two people have knowledge about a feature, compared to when all team members work on their own separate tasks. This lowers the potential impact if one team member should become available, as at least one other person have knowledge of the task. This measure also contributes to implicit knowledge sharing, as team members inherently will collaborate more than they would on unrelated tasks. This was also stated to improve team member satisfaction, and that it was more enjoyable to work together with someone on a task than working alone.

### 5.1.2 What hinders effective teamwork?

The second part of the first research question of this thesis was given as follows:

> **RQ1.2:** What hinders effective teamwork?

The most commonly mentioned factors that hinder effective teamwork are shown in Table 10, sorted by prevalence.

Table 10: Factors that hinder effective teamwork

| Factor | Participants |
|---|---|
| Remote work | P5, P6, P7, P8, P13, P14 |
| Misalignment | P7, P8, P9, P12 |
| Lack of documentation | P2, P6, P13, P14 |
| Lack of team spirit | P6, P7, P8, P10 |
| Inter-team requests | P2, P5 |
| Task dependencies | P7, P8 |

**Remote work**

While several of the participants were very pleased with being able to work remotely, there has also been raised a significant number of concerns regarding its impact on teamwork. The most commonly mentioned challenge with hybrid teams was that those working from home could become isolated and miss out on relevant discussions (P7, P8, P14). Some attempts to solve this issue have been tried, such as documenting the results of discussions online or inviting those that are not present to a digital meeting, but documenting everything has proved difficult and spontaneous discussions are natural and hard to avoid.

Hybrid work can also be challenging for those present at the office, as they may need to sit in meeting rooms when joining online meetings to avoid disturbing others, which may be a limited resource (P13). Lastly, some of the participants have expressed that digital communication is simply inferior to talking to someone in person (P5, P6). This is also consistent with the Agile Manifesto, which claims that face-to-face conversations are the most effective method of communication in development teams (Beck, Beedle et al. 2001). Arranging online meetings can take more time than just nudging someone at the same office, especially for small requests or simple questions. Additionally, context or details may be lost if you need to write everything down in a post or message, rather than showing someone the issue in person.

### Misalignment

In general, there were two main types of misalignment that were found to hinder effective teamwork, goal misalignment and process misalignment. As for goal misalignment, this was found both on the inter-team and intra-team levels. Without a firm team goal for the product that is being built, various team members could develop their own personal goals for the product, which could be counter-productive. Similarly on the inter-team level, without a common goal, teams could end up working against each other instead of pulling in the same direction. When it comes to process misalignment, this is related to a high level of autonomy. When teams are allowed to determine their own work process, they tend to end up with a process that is customized to their own specific team.

However, when there are a lot of differences between the work processes of teams, collaboration across teams can be more complicated. One example is when the sprints of two teams are not aligned. One team may discover that they are blocked by the other team during their sprint planning, and request help from the other team. However, as the other team does not have sprint planning at the same time, they may need to wait until their planning session to be able to prioritize the request and add it to the sprint backlog. Although autonomy was appreciated, there was also a need for alignment across teams. The same was also found in the study of a large-scale program conducted by Berntzen et al. (2021).

### Lack of documentation

Some participants expressed that their team could benefit from having more extensive documentation of their systems and products. With sufficient documentation, there are more tasks that an individual can accomplish on their own rather than requesting help from others. A lack of documentation can hurt teamwork effectiveness by increasing the amount of collaboration and coordination that is needed, which ties up resources. However, it was also noted that too much documentation could be overwhelming. In order for the documentation to provide value, it needs to be manageable and understandable, and it needs to be simple to navigate to find the parts that are needed.

### Lack of team spirit

Good team spirit was seen as an important factor for maintaining effective teamwork by several of the participants. Hence, lacking team spirit can hurt teamwork effectiveness in teams. One participant highlighted that while there was good team spirit within teams, team members could benefit from more bonding with those outside the team (P6). Lack of inter-team bonding could be connected with inter-team coordination challenges, which were commonly found. One aspect that can hurt team spirit is changes to the members of a team, as some participants expressed that team spirit is built up over time with a stable team. Further, budget cuts were also mentioned as an aspect that could impact teams and lower the motivation of team members.

### Inter-team requests

Both Edison et al. (2022) and Dikert et al. (2016) have found inter-team coordination to be one of the main categories of challenges reported in papers on large-scale agile software development and large-scale agile transformations. The same was also found in this research. Several of the participants expressed that there could be a lot of inter-team dependencies, that lead to inter-team requests. This may be requests for tasks to be done, help with an issue, or for information that is required to progress. While teams generally are happy to help other teams when needed, too many of these requests can make a team overwhelmed with extra work, inhibiting them from progressing on their own tasks. This was a bigger issue for teams that had a lot of experienced team members, as they were more likely to be asked for help.

### Task dependencies

Dependencies were not only found on the inter-team level but also within teams. When developing a new feature, such features were often broken up into smaller tasks. However, these tasks could sometimes depend on each other, meaning that they needed to be completed in a specific order. This also prevented multiple developers from working on multiple of these tasks in parallel. This could lead to waiting and blocking, as one developer was unable to progress until another had fin-

ished their task. Dependencies between tasks are therefore something that should be identified during planning. One solution could be to plan dependent tasks for separate sprints, ensuring that they are undertaken in the right order. Still, task dependencies may not always be trivial to identify before starting to work on them.

## 5.2 Enabling effective teamwork

The previous section displayed which factors have been found to impact teamwork effectiveness in large-scale agile development, either in a positive or a negative way. But what can large-scale agile teams do to improve their chances of effective teamwork? This section will recommend measures that have been used and appreciated by the interviewed teams. It will hence attempt to answer the second research question, which was given as follows:

> **RQ2:** How do large-scale agile teams enable teamwork effectiveness?

### 5.2.1 Customize agile methods

Since the inception of the agile philosophy, a multitude of various agile methods have been developed to help development teams work in an agile way. From the interviews, it was revealed that the most popular agile methods in the teams were Scrum, Kanban, and XP, while the Signicat case also used some aspects of the large-scale agile framework the Spotify model. While some of these methods may seem prescriptive, some believe that they should not be used as an exact blueprint and that every team should customize its own methods (Kniberg 2015). This thought was also shared by several of the participants, amongst them a developer from Signicat who expressed that agile teams should drop the parts from a method that does not suit the team (P4). They also stated that customizing methods can increase both the happiness and efficiency of an agile team.

The members of a team know best what works for their team, and being allowed to develop your own work process may be more motivating than being forced to follow a specific template. In other words, the agile methods should be customized to the

team, not the other way around. Additionally, when adjusting your agile process, the help of Scrum Masters and agile coaches has been greatly appreciated. One reason could be that while all team members may be allowed to suggest changes to the team's process, people may feel that the threshold for making changes can be too high when it is not their explicit responsibility. By appointing a Scrum Master, it can be easier for that team member to give directions to the rest of the team regarding the agile process, as this is expected from them. It was also highlighted that giving the Scrum Master responsibility to someone who is motivated for the task was helpful. External agile coaches were also used in NAV IT and these were also highly appreciated. While coaches from outside the team may not have as detailed knowledge of the team, they could bring in knowledge of what has worked in other teams, which could be helpful to improve the agile process.

### 5.2.2 Share your knowledge

The knowledge of employees is an essential asset in software development companies. This knowledge is built up over time and consists of technical knowledge, domain knowledge, and knowledge of the software systems that are being built. However, if team members move on from their team or from their company, the knowledge that they had may be lost if knowledge-sharing techniques are not employed. Pair programming is one knowledge-sharing practice that could be used, and which was commonly used in NAV IT. Participants from this case also had some recommendations for reaping the biggest benefits of this technique. Pairing up experienced and inexperienced developers together was seen as beneficial to increase the amount of knowledge that is shared. Additionally, it was recommended to identify which areas team members are interested learn and let them participate in pair programming on matters they were interested in, to further improve the chances of successful knowledge sharing.

Another measure that was deemed effective regarding knowledge sharing was assigning multiple developers to similar tasks. This leads to implicit knowledge sharing, as colleagues typically need to cooperate to complete their tasks. This can improve teamwork, by ensuring that people are cooperating rather than working isolated.

It was also seen as a good measure for improving team member satisfaction, as developers preferred collaborating with others over working alone.

### 5.2.3 Reflect to improve

Reflection at regular intervals, in order to get more effective as a team, is one of the principles of agile (Bjørnson and Vestues 2016). Practices for sharing reflection with each other were something that all the interviewed teams used, and reflection was perceived as important for team improvement by several of the participants. Retrospective meetings were the most common practice for reflection, which all the teams used in some form. Some might believe that spending the valuable time of a whole team to analyze the past might be wasteful. For teams following a lean methodology, eliminating waste is one of the fundamental principles, and meetings that do not directly generate value could be perceived as waste. However, multiple of the participants highlighted that retrospective meetings were a good use of time. Reflection is seen as an important part of being able to learn from mistakes and improve as a team. Retrospective meetings were also one of the main arenas used for feedback within the teams. Feedback was also perceived as vital for improving teamwork, and enabling team members to share their feedback by hosting reflection sessions may therefore be key.

### 5.2.4 Give considerable control to teams

Another principle from the agile manifesto states that *"The best architectures, requirements, and designs emerge from self-organizing teams."* (Beck, Beedle et al. 2001). Allowing teams to have autonomy and allowing them to make big decisions can have a positive impact both on the quality of the product being built, but also employee satisfaction. Further, a lack of team autonomy has been found to be a key challenge in large-scale agile development (Edison et al. 2022). In general, the participants from the interviews believed that being trusted by higher management to make certain decisions themselves was motivating. They also felt that by deciding their own work process, they could create a customized process that was as well

suited to their team as possible. Also, by allowing developers a say in which tasks they want to work on, satisfaction improves (P2). However, it was highlighted that experienced individuals could take up too much space in autonomous teams. This can make those less experienced more reluctant to share their opinions and take on responsibility, which may limit their growth and learning. Hence, team members of all experience levels should be encouraged to contribute to decision-making and dominating individuals should be encouraged to make space for others. Additionally, some guidelines should be developed by upper management and followed by teams, to alleviate inter-team coordination. Such guidelines could for example be specific metrics for monitoring progress, lines of communication, or interfaces between sub-systems.

### 5.2.5 Get to know your colleagues

Not knowing and/or trusting colleagues was perceived as one of the key reasons for ineffective teamwork. This could be connected with the finding that coordination is significantly harder across teams than within teams. However, getting to know your teammates could also be beneficial, as it increases team spirit which in turn is positive for teamwork. Social gatherings and initiatives have been highlighted as beneficial for teams, but work-related interactions can also create familiarity between colleagues. Both physical and digital gatherings can be helpful, but several participants expressed that digital interactions were not as effective as meeting face-to-face.

As Signicat has employees located in several various countries, some teams consist of team members who are not colocated. In one of these teams, the majority of the team members were located in Norway, while two developers were situated in Portugal. A measure that was beneficial for this team was to invite those living in Portugal to gather in Norway with the rest of the team for a few days and get to know their colleagues better. This was seen as helpful for future teamwork in the team, which suggests that knowing someone on a personal level is beneficial also for digital collaboration. Cross-site visits are also something Kniberg (2015) recommends for distributed teams.

## 5.3 Comparison to an existing model

Some models on teamwork effectiveness have been developed in the past, such as the *agile teamwork effectiveness model* (ATEM). This model is developed specifically for agile teams, as a revision of the more general teamwork effectiveness model, the Big Five model (Salas et al. 2005; D. Strode, Dingsøyr et al. 2022). As a basis for this comparison, ATEM is chosen over the Big Five model as it considers agile teams which is closer to the context of the cases in this study. So how do the findings of this study fit in with this existing model? Is ATEM a valid model for large-scale agile as well, or are there other factors that impact teamwork specifically in the large-scale context? This section will compare the findings of this study to the five teamwork core components and three coordinating mechanisms of ATEM, and either confirm or deny that they are supported by the statements of the participants. Further, any additional components or coordinating mechanisms that are found will be proposed as an extension or adjustment of the model for the large-scale agile software development context. Thus, the section will attempt to answer the third and final research question, which was given as follows:

> **RQ3:** How do the findings compare to the existing model on teamwork effectiveness, ATEM?

### 5.3.1 Coordinating mechanisms

Coordinating mechanisms are mechanisms that support the teamwork components, and which are needed to extract value from each of the components (Salas et al. 2005). They are also described as facilitating the teamwork components (D. Strode, Dingsøyr et al. 2022). The three coordinating mechanisms from ATEM are *shared mental models*, *mutual trust*, and *communication*.

#### Shared mental models

The concept of shared mental models represents a common understanding between team members of goals and tasks which may be necessary to enable effective team-

work. When a shared mental model is established, team members can be able to anticipate each other's needs and adjust work strategies to adapt to changes (D. Strode, Dingsøyr et al. 2022). Shared mental models were commonly found in the interviewed teams, with some of the participants expressing that their team relied heavily on shared understandings and knowledge. The type of shared mental model that was most commonly highlighted was a shared understanding of how their products were built up, with all its sub-systems and components. This includes a thorough technical understanding of how different systems are integrated with each other, which components are responsible for what, and in general how to maintain and further develop the software. It was shared that such an understanding was hard to document in its entirety, and the shared mental models were therefore important for the teams in order to be able to progress. Another type of shared mental model that was brought up was to have a common language or jargon (P3, P6). This includes the naming of components and concepts within a project, and this common way of communicating was helpful for avoiding misunderstandings and improving the effectiveness of communication. As such, shared mental models are supported as a coordinating mechanism by the interview data.

**Mutual trust**

Trusting those that you collaborate with is an integral part of most types of teamwork, and this includes agile development teams. It is also a part of one of the principles of agile, which states: *"Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."* (Beck, Beedle et al. 2001). Several of the participants noted that teamwork was more difficult the level of trust was low between those collaborating. This was especially apparent in the inter-team context, where trust levels were generally lower than within teams. In general, the aspect of mutual trust was not something that was brought up very frequently during the interviews, but those who did generally expressed that there was a high level of trust between team members in their team. However, as low levels of trust were highlighted as hindering teamwork effectiveness, mutual trust is supported as a coordinating mechanism.

#### Communication

Communication is a vital aspect of teamwork which is required in some form to be able to collaborate with others. In the software development field, both written and verbal communication is frequently used to coordinate efforts. Some examples of written communication are online messages, documentation, and plans, while verbal communication typically occurs in group meetings or one-to-one discussions. The creators of ATEM also argue that communication is an important aspect, and state that it is *"key to efficient software development in agile teams"* (D. Strode, Dingsøyr et al. 2022). They also found that group communication, like for example meetings, were more common for agile teams than one-to-one communication. In this study, both examples of sender-to-receiver and sender-to-group communication were found. Some teams had daily stand-up meetings, and as a consequence were able to do a lot of the required coordination as a group. However, other teams seldom used common team meetings. The participants from these teams generally expressed that communication was primarily needed when they were stuck on a task. In these cases, they often reached out to other developers that they knew could help in the form of sender-to-receiver communication. Still, the topic of communication was often brought up as a critical element of functioning as a team. The general consensus from the participants was that effective communication was paramount to achieving effective teamwork. Therefore, communication is supported as a coordinating mechanism.

### 5.3.2 Core components

The core components are the five elements that make up the "Big Five", and which are suggested to promote team effectiveness (Salas et al. 2005). These core components have been revised for ATEM, resulting in the following five components: *shared leadership*, *peer feedback*, *redundancy*, *adaptability*, and *team orientation* (D. Strode, Dingsøyr et al. 2022).

#### Shared leadership

From their research of agile teams, D. Strode, Dingsøyr et al. (2022) found that in

agile contexts, teams have a shared form of leadership rather than a single designated leader. Shared leadership was also a concept that was found in the participating teams. Several of the teams had a product owner, who was responsible for prioritizing tasks, or an architect who was responsible for technical architectural decisions. However, those who had extra responsibilities were open to suggestions from other team members, and it was not perceived that individuals had significantly more leadership influence than the average team member. Several of the participants expressed that their team had a high level of autonomy, and being self-organized was seen as positive for team performance and team member satisfaction. As such, shared leadership is supported as a core component for teamwork effectiveness.

**Redundancy**

Redundancy in this context is related to several team members possessing overlapping skill sets so that multiple team members are able to perform the same task. A redundancy in skill sets was something that several participants expressed that their teams tried to achieve. It was stated that this was a risk-reducing measure and that it could enable teams to be more versatile. Even though achieving competence redundancy was a goal for the teams, it was stated that it was something that was hard to reach. The main reasons for this were limited resources, both in the form of too little rest capacity to expand the skill sets of existing team members, but also a lack of budget to hire more developers. Still, it was expressed that redundancy could make teams more robust and able to maintain team performance even in challenging situations, for example when certain team members are unavailable. This stability and versatility make collaboration more predictable, which boosts teamwork. Therefore, redundancy is supported as a core component.

**Adaptability**

The software development field can be an unpredictable world, where technology advance quickly and user requirements can be unclear and ever-changing. This unpredictable environment calls for adaptable development teams, which is also one of the reasons for the popularity of agile methods. Adaptability encompasses being able to change direction and make changes as a team, while still maintaining

a satisfactory level of team performance. Most of the participants expressed that their teams were highly adaptable and were able to respond quickly to changes. Being able to re-organize and re-assign tasks is one aspect that has been highlighted as important for team performance (P1). In order to be able to re-arrange tasks effectively, a high level of adaptability is needed. In general, staying versatile as a team was seen as a measure that lowers the risk of unproductivity, and that is needed in the work environment of the agile teams. Hence, adaptability is supported as a core component.

**Team orientation**

The concept of team orientation relates to the degree to which team members act as team players. This includes working towards the goals of the team, but it also encompasses factors such as team spirit and sense of community. Team spirit is something that in general has been at a high level in the interviewed teams. Team orientation on the team level is a concept that was recognized by the participants, even in multi-team settings. One proof of team orientation that was frequently found in the teams was that team members were happy to offer their help to others even though they might have enough work on their plate already. Additionally, some also stated that people were more concerned with getting things done than doing things their way. The importance of team orientation was also highlighted, with one developer stating that good team spirit correlated to better performances, higher satisfaction, and overall better health (P13). It was also noted that having a common identity and sense of community in the team made all tasks easier to accomplish (P14). As such, team orientation is supported as a core component.

### 5.3.3 Additional components or mechanisms

Supporting evidence was found for all three coordinating mechanisms and all five core components of ATEM. This contributes to further validating the model for agile teams. It also shows that ATEM can be applicable to agile teams in the large-scale context as well. But could the model be extended with other components that are found in large-scale agile development?

### Knowledge sharing

Knowledge-sharing is present in ATEM within the core components of "redundancy" and "shared mental models". However, as knowledge sharing as a concept has been brought up frequently in the interviews, this study suggests that it could be defined as a separate coordinating mechanism. Knowledge sharing impacts and supports several of the existing core components, primarily "redundancy" and "adaptability". Improving competence redundancy can generally be achieved in one of two ways, either by hiring additional developers with the right competence or training existing team members. As many of the interviewed teams deal with complex, bespoke systems, knowledge sharing between experienced and inexperienced developers has been seen as potentially the most effective way of improving the competence of team members. In the large-scale agile software development world in general, it is reasonable to assume that many programs deal with large, complex systems that have been built over an extensive time period. When dealing with these types of systems it is also reasonable to believe that those most qualified to train newcomers are those who have worked on the system for some time.

Knowledge sharing has also been labeled as "risk-reducing" by participants, referring to a lower risk of sudden team performance loss. This is related to adaptability, as sufficient knowledge sharing can make the team more equipped to deal with changes and uncertainties. Whether knowledge sharing is more merited to being a coordinating mechanism in the large-scale context compared to traditional agile development may not be trivial to determine. One could assume that with more teams and people involved, systems grow more complex, and more information is generated which should be managed correctly. As more people are involved, higher standards of communication, as well as management and distribution of knowledge may be required. Hence, assuming that effective knowledge sharing is more important for large-scale agile teams is reasonable.

### Colocation

Colocation is also already present in ATEM, as it is a sub-component of the coordinating mechanism "communication". The creators of ATEM found that colocation

and physical presence fostered team effectiveness, while a lack thereof was a hindering factor. From the interviews in this study, the use of remote and hybrid work environments, and consequently the degree of colocation, has been a frequently brought up topic. So much so that the findings suggest that colocation could constitute its own core component of teamwork effectiveness. While some have highlighted the benefits of remote work, such as increased focus, several challenges have been brought up related to not being colocated. Several participants revealed that in a hybrid work environment, those working from home could become isolated from those working from the office. This includes missing out on ad-hoc discussions and typical workplace social interactions. It was also mentioned that digital communication can be less effective than face-to-face interactions, even with all the digital communication tools that are available today.

The degree of colocation does not only impact communication, however, as it could impact other psychological aspects as well, like for example the sense of community of a team. This could suggest that moving colocation out of the communication component could be appropriate. The finding that colocation impacts teamwork effectiveness is also consistent with existing literature, such as Henrik Kniberg's "Scrum and XP from the Trenches" (Kniberg 2015). Kniberg states that while distributed teams are common and could be successful, being physically colocated increases the productivity of agile teams. But does the scale of agile projects impact the effects of colocation? One interesting point is that the case where developers worked remotely by choice faced fewer challenges with not being colocated than the case that had to work distributed because team members were geographically separated. Further, it could be reasonable to assume that as companies grow larger, they expand to different geographical locations which could increase the likelihood of distributed teams. Consequently, colocation could be a more important component in large-scale agile than in single-team projects.

The resulting extension of ATEM, the large-scale agile teamwork effectiveness model (LATEM), with its four coordinating mechanisms and six core components, is visually represented in Figure 16.

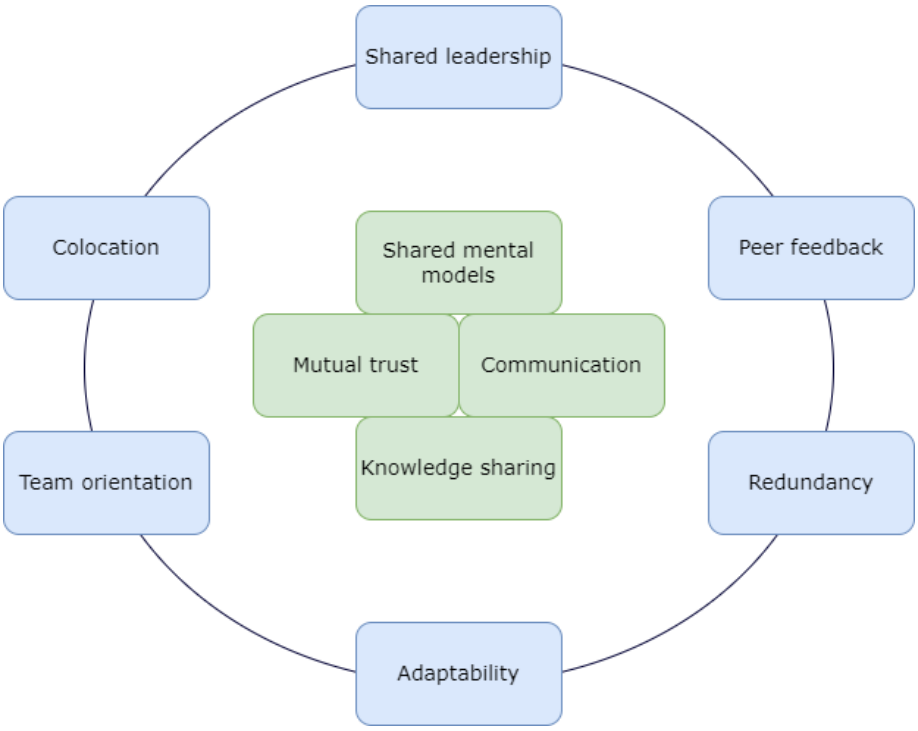



Figure 16: LATEM

## 5.4 Evaluation and limitations

This study has researched teamwork effectiveness in two cases that employ agile methods at a scale. However, the fact that only two cases are involved limits the number of perspectives on the topic. While there is no "right" number of cases to involve in a multi-case study, the impact of sample size in qualitative research has been previously explored. In the study of Marshall et al. (2013), 83 qualitative studies from the information systems field were examined. They found that while almost a third of multi-case studies consisted of two cases, the median was five cases. Moreover, the data from this study comes from a total of 14 interviews. While this number of interviews satisfied the target set from the research design phase, additional interviews could further strengthen the findings. In the case of NAV IT, all the participants came from the same product area. While this gave a detailed insight into the current state of this area of the organization, perspectives from multiple product areas could have given a more accurate representation of NAV IT as a whole.

Lastly, both of the cases have gone through large agile transformations and organization restructuring fairly recently. NAV IT transitioned from outsourcing to bringing the development and maintenance of its IT systems in-house in 2017. At the same time, they also underwent an agile transformation. While this is six years ago, the organization is still in a process of change. For instance, several participants noted that a new team setup was about to be implemented at the time of the interviews. In the case of Signicat, an organizational restructuring took place in 2021, where structures such as tribes and guilds were introduced. This transformation is even more recent, therefore, it is reasonable to assume that a complete status quo may not have been reached yet. As large transformations are relatively recent in both these cases, and some changes may still be underway, findings about teamwork in these cases may become outdated quickly. Hence, examining cases that have employed agile at scale for a longer period of time may give more accurate results on teamwork in this context.

# 6 Conclusion

This study has researched several aspects of teamwork effectiveness in large-scale agile software development. To enable this research, two software development organizations were researched through the use of semi-structured interviews. Data from these interviews were analyzed and used to answer the research questions that were to be investigated. Firstly, the factors that impact teamwork effectiveness in the large-scale agile context were identified. The following factors were found to foster teamwork effectiveness: *right agile methods*, *knowledge sharing techniques*, *Scrum Master/agile coach*, *digital tools*, *feedback*, *remote work*, and *working on the same task*. As these seven factors were found to improve the effectiveness of teamwork, they may be used by other large-scale agile projects to attempt to improve the chances of effective teamwork. Further, the following factors were found to hinder teamwork effectiveness: *remote work*, *misalignment*, *lack of documentation*, *lack of team spirit*, *inter-team requests*, *task dependencies*. Hence, minimizing the prevalence of these factors can help large-scale agile projects improve their teamwork effectiveness. Interestingly, the factor "remote work" was found both to foster and hinder teamwork effectiveness. As such, it is unclear whether making use of distributed teams improves or decreased teamwork effectiveness.

Secondly, techniques and advice for enabling teamwork effectiveness were found. These strategies had been used in the researched cases and were credited with improving collaboration, easing teamwork, and improving productivity. The identified techniques and advice were *customize agile methods*, *share your knowledge*, *reflect to improve*, *give considerable control to teams*, and *get to know your colleagues*. By following this advice, large-scale agile projects can set themselves up for teamwork success, according to the research data.

Lastly, the results from the two large-scale agile cases have been compared to the existing model of agile teamwork effectiveness, ATEM. The findings supported all five teamwork components and three coordinating mechanisms from ATEM, suggesting that these elements also impact teamwork in a large-scale context. Further, one new teamwork component and one new coordinating mechanism were suggested

specifically for effective teamwork in agile at scale. *Colocation* was suggested as a new teamwork component due to how frequently it was brought up as a factor during the interviews. It was also found that colocation impacts other facets of teamwork than solely communication, leading to the suggestion of separating it into a new component. The component of colocation was assumed to be more important in large-scale programs, as distributed teams may be more common in larger projects. Moreover, *knowledge sharing* was suggested as a new coordinating mechanism for the large-scale context. Knowledge-sharing techniques were found to be a key enabler for several of the teamwork components and knowledge-sharing was found to be important to achieve high levels of teamwork effectiveness. The mechanism of knowledge sharing was assumed to be more important in large-scale programs, as they can entail more knowledge, more people, and more complex systems.

## 6.1 Contributions

This study provides several contributions, both to the agile software development field and to practitioners. Firstly, it provides empirical research on the topic of teamwork in the large-scale agile software development context. More empirical research on teamwork and team effectiveness is something that Dingsøyr and Dybå (2012) have called for. They also expressed a need for testing theories from other fields, and adjusting them to software teams. This study does that, by comparing findings from a large-scale agile context with the existing teamwork model for agile teams, ATEM. Additional empirical research of agile at scale is also something that the field of agile software development has called for, as "Agile and large projects" was voted the top burning question at the 2010 XP conference (Freudenberg and Sharp 2010). Further, the study provides a contribution in the form of identified factors that impact teamwork effectiveness and prescriptive advice that can be of use to practitioners. Lastly, this study provides a basis that can be used for further research on teamwork in large, agile projects.

## 6.2 Future work

This study has given some insight into teamwork in large-scale agile software development, but there are several kinds of research that could further what has already been found on this topic. Some of the shortcomings of this study reveal possible advancements that can be made. Due to the time restrictions imposed on this study, long-term effects could not be investigated. As such, the findings on changes in teamwork effectiveness relied on the memories of the interview participants. Hence, a longitudinal study on long-term changes in teamwork effectiveness in a large-scale project could give further insight, as a researcher would be able to observe changes themselves. This study also relied solely on one data generation method, namely interviews. A study on teamwork in large-scale agile using multiple data generation methods could yield more accurate results due to the use of several types of sources. Specifically, the use of observations may be beneficial in this type of research, as it gives the opportunity to assess teamwork in a more natural setting.

Moreover, one ambiguity of this study is whether or not remote and hybrid work environments are beneficial to teamwork effectiveness. While some participants had nothing but positive experiences with working remotely, others expressed that it impacted teamwork negatively. As such, this is an area that could benefit from further research. This is also a relevant area of study as more and more people have started working remotely in recent times (Felstead and Henseke 2017). Another potential route to further the research is to investigate how teamwork effectiveness impact productivity. It is reasonable to assume that a high level of productivity is important to most companies, so investigating possible connections to teamwork may be worthwhile.

# Bibliography


Babb, J, R Hoda and J Nørbjerg (2014). 'Embedding Reflection and Learning into Agile Software Development'. In: *IEEE Software* 31.4, pp. 51–57. ISSN: 1937-4194. DOI: 10.1109/MS.2014.54.

Bailey, Diane and Nancy Kurland (May 2002). 'A Review of Telework Research: Findings, New Directions, and Lessons for the Study of Modern Work'. In: *Journal of Organizational Behavior* 23, pp. 383–400. DOI: 10.1002/job.144.

Beck, Kent and Cynthia Andres (2004). *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional. ISBN: 0321278658.

Beck, Kent, Mike Beedle et al. (2001). *Manifesto for Agile Software Development*. URL: http://www.agilemanifesto.org/.

Berntzen, Marthe, Viktoria Stray and Nils Brede Moe (2021). 'Coordination Strategies: Managing Inter-team Coordination Challenges in Large-Scale Agile'. In: *Agile Processes in Software Engineering and Extreme Programming*. Ed. by Peggy Gregory et al. Cham: Springer International Publishing, pp. 140–156. ISBN: 978-3-030-78098-2.

Bjørnson, Finn Olav and Torgeir Dingsøyr (2008). 'Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used'. In: *Information and Software Technology* 50.11, pp. 1055–1068. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2008.03.006. URL: https://www.sciencedirect.com/science/article/pii/S0950584908000487.

Bjørnson, Finn Olav and Kathrine Vestues (2016). 'Knowledge Sharing and Process Improvement in Large-Scale Agile Development'. In: *Proceedings of the Scientific Workshop Proceedings of XP2016*. XP '16 Workshops. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450341349. DOI: 10.1145/2962695.2962702. URL: https://doi.org/10.1145/2962695.2962702.

Bourque, Pierre, Richard E Fairley and IEEE Computer Society (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. 3rd. Washington, DC, USA: IEEE Computer Society Press. ISBN: 0769551661.

Conboy, Kieran and Noel Carroll (Mar. 2019). 'Implementing Large-Scale Agile Frameworks: Challenges and Recommendations'. In: *IEEE Software*. DOI: 10.1109/MS.2018.2884865.

Daljajev, Kadri et al. (2020). 'A Study of the Agile Coach's Role'. In: *Product-Focused Software Process Improvement*. Ed. by Maurizio Morisio, Marco Torchiano and Andreas Jedlitschka. Cham: Springer International Publishing, pp. 37–52. ISBN: 978-3-030-64148-1.

DeSanctis, Gerardine (1984). 'Attitudes toward telecommuting: Implications for work-at-home programs'. In: *Information & Management* 7.3, pp. 133–139. ISSN: 0378-7206. DOI: https://doi.org/10.1016/0378-7206(84)90041-7. URL: https://www.sciencedirect.com/science/article/pii/0378720684900417.

Deshpande, Advait et al. (2016). 'Remote Working and Collaboration in Agile Teams'. In: *International Conference on Interaction Sciences*.

Digital.ai (2021). *15th State of Agile Report*. Tech. rep.

— (2022). *16th State of Agile Report*. Tech. rep. Boston.

Dikert, Kim, Maria Paasivaara and Casper Lassenius (2016). 'Challenges and success factors for large-scale agile transformations: A systematic literature review'. In: *Journal of Systems and Software* 119, pp. 87–108. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2016.06.013. URL: https://www.sciencedirect.com/science/article/pii/S0164121216300826.

Dingsøyr, Torgeir and Tore Dybå (Mar. 2012). 'Team effectiveness in software development: Human and cooperative aspects in team effectiveness models and priorities for future studies'. In: *2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering, CHASE 2012 - Proceedings*. DOI: 10.1109/CHASE.2012.6223016.

Dingsøyr, Torgeir, Tor Fægri and Juha Itkonen (Dec. 2014). *What Is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development*. Vol. 8892. ISBN: 978-3-319-13834-3. DOI: 10.1007/978-3-319-13835-0{_}20.

Dingsøyr, Torgeir, Davide Falessi and Ken Power (2019). 'Agile Development at Scale: The Next Frontier'. In: *IEEE Software* 36.2, pp. 30–38. DOI: 10.1109/MS.2018.2884884.

Dingsøyr, Torgeir and Nils Brede Moe (Aug. 2013). 'Research Challenges in Large-Scale Agile Software Development'. In: *SIGSOFT Softw. Eng. Notes* 38.5, pp. 38–39. ISSN: 0163-5948. DOI: 10.1145/2507288.2507322. URL: https://doi.org/10.1145/2507288.2507322.

— (2014). 'Towards Principles of Large-Scale Agile Development'. In: *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*. Ed. by Torgeir Dingsøyr et al. Cham: Springer International Publishing, pp. 1–8. ISBN: 978-3-319-14358-3.

Dingsøyr, Torgeir, Nils Brede Moe and Eva Amdahl Seim (Oct. 2018). 'Coordinating Knowledge Work in Multiteam Programs: Findings From a Large-Scale Agile Development Program'. In: *Project Management Journal* 49.6, pp. 64–77. ISSN: 8756-9728. DOI: 10.1177/8756972818798980. URL: https://doi.org/10.1177/8756972818798980.

Dybå, Tore and Torgeir Dingsøyr (Apr. 2009). 'What Do We Know about Agile Software Development?' In: *Software, IEEE* 26, pp. 6–9. DOI: 10.1109/MS.2009.145.

Earl, Michael (2001). 'Knowledge Management Strategies: Toward a Taxonomy'. In: *Journal of Management Information Systems* 18.1, pp. 215–233. ISSN: 07421222. URL: http://www.jstor.org/stable/40398522.

Edison, Henry, Xiaofeng Wang and Kieran Conboy (2022). 'Comparing Methods for Large-Scale Agile Software Development: A Systematic Literature Review'. In: *IEEE Transactions on Software Engineering* 48.8, pp. 2709–2731. DOI: 10.1109/TSE.2021.3069039.

Felstead, Alan and Golo Henseke (May 2017). 'Assessing the growth of remote working and its consequences for effort, well-being and work-life balance'. In: *New Technology, Work and Employment* 32. DOI: 10.1111/ntwe.12097.

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

Fuchs, Christoph and Thomas Hess (Nov. 2018). 'Becoming Agile in the Digital Transformation: The Process of a Large-Scale Agile Transformation'. In.

Girma, Melaku, Nuno M. Garcia and Mesfin Kifle (May 2019). 'Agile Scrum Scaling Practices for Large Scale Software Development'. In: *2019 4th International Conference on Information Systems Engineering (ICISE)*. IEEE, pp. 34–38. ISBN: 978-1-7281-2558-9. DOI: 10.1109/ICISE.2019.00014.

Gloet, Marianne and Milé Terziovski (Jan. 2004). 'Exploring the relationship between knowledge management practices and innovation performance'. In: *Journal of Manufacturing Technology Management* 15.5, pp. 402–409. ISSN: 1741-038X. DOI: 10.1108/17410380410540390. URL: https://doi.org/10.1108/17410380410540390.

Hansen, Morten T, N Nohria and Tom Tierney (1999). 'What's your strategy for managing knowledge?' In: *Harvard business review* 77 2, pp. 106–16.

Hanssen, Geir Kjetil, Tor Stlhane and Thor Myklebust (2018). *SafeScrum Agile Development of Safety-Critical Software*. 1st. Springer Publishing Company, Incorporated. ISBN: 331999333X.

Henrik Kniberg (2011). *Lean from the Trenches: Managing Large-scale Projects with Kanban*. Pragmatic Bookshelf.

Hoegl, Martin, bullet Hans and Hans Gemuenden (Mar. 2001). 'Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence'. In: *INFORMS* 12, pp. 435–449. DOI: 10.1287/orsc.12.4.435.10635.

Ian Sommerville (2010). *Software Engineering*. 9th ed.

Ingvaldsen, Jonas A and Monica Rolfsen (June 2012). 'Autonomous work groups and the challenge of inter-group coordination'. In: *Human Relations* 65.7, pp. 861–881. ISSN: 0018-7267. DOI: 10.1177/0018726712448203. URL: https://doi.org/10.1177/0018726712448203.

Jørgensen, Magne (2018). 'Do Agile Methods Work for Large Software Projects?' In: *Agile Processes in Software Engineering and Extreme Programming*. Ed. by Juan Garbajosa, Xiaofeng Wang and Ademar Aguiar. Cham: Springer International Publishing, pp. 179–190. ISBN: 978-3-319-91602-6.

Klopotek, Magdalena (May 2017). 'The Advantages and Disadvantages of Working Remotely from the Perspective of Young Employees'. In: Management Challenges in a Network Economy: Proceedings of the MakeLearn and TIIM International Conference 2017. ToKnowPress, p. 535. URL: https://ideas.repec.org/h/tkp/mklp17/535.html.

Kniberg, Henrik (2010). *Kanban and Scrum - Making the Most of Both*. Lulu.com. ISBN: 0557138329.

— (2015). *Scrum and XP from the Trenches: 2nd Edition*. Lulu.com. ISBN: 1430322640.

Kniberg, Henrik and Anders Ivarsson (2012). *Scaling Agile @ Spotify*. URL: https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf.

Kotlarsky, Julia and Ilan Oshri (Mar. 2005). 'Social ties, knowledge sharing and successful collaboration in globally distributed system development projects'. In: *European Journal of Information Systems* 14.1, pp. 37–48. ISSN: 0960-085X. DOI: 10.1057/palgrave.ejis.3000520. URL: https://doi.org/10.1057/palgrave.ejis.3000520.

Kozlowski, Steve and Bradford Bell (May 2003). 'Work Groups and Teams in Organizations'. In: *Articles & Chapters* 14. DOI: 10.1002/0471264385.wei1214.

Kraut, Robert E and Lynn A Streeter (Mar. 1995). 'Coordination in Software Development'. In: *Commun. ACM* 38.3, pp. 69–81. ISSN: 0001-0782. DOI: 10.1145/203330.203345. URL: https://doi.org/10.1145/203330.203345.

Langfred, Claus W (2007). 'The Downside of Self-Management: A Longitudinal Study of the Effects of Conflict on Trust, Autonomy, and Task Interdependence in Self-Managing Teams'. In: *The Academy of Management Journal* 50.4, pp. 885–900. ISSN: 00014273. URL: http://www.jstor.org/stable/20159895.

Malone, T W and K Crowston (Mar. 1994). 'The Interdisciplinary Study Of Coordination'. In: *ACM COMPUTING SURVEYS* 26.1, pp. 87–119. ISSN: 0360-0300. DOI: 10.1145/174666.174668.

March, J.G. and H.A. Simon (1958). *Organizations*. New York: Wiley.

Marshall, Bryan et al. (Sept. 2013). 'Does Sample Size Matter in Qualitative Research?: A Review of Qualitative Interviews in is Research'. In: *Journal of Computer Information Systems* 54, pp. 11–22. DOI: 10.1080/08874417.2013.11645667.

Micaela, B (2020). 'Telework in the EU before and after the COVID-19: where we were, where we head to'. In: URL: https://joint-research-centre.ec.europa.eu/system/files/2021-06/jrc120945_policy_brief_-_covid_and_telework_final.pdf.

Moe, Nils, Helena Olsson and Torgeir Dingsøyr (May 2016). *Trends in Large-Scale Agile Development: A Summary of the 4th Workshop at XP2016*. DOI: 10.1145/2962695.2962696.

Mohagheghi, Parastoo and Magne Jørgensen (2017). 'What Contributes to the Success of IT Projects? Success Factors, Challenges and Lessons Learned from an Empirical Study of Software Projects in the Norwegian Public Sector'. In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pp. 371–373. DOI: 10.1109/ICSE-C.2017.146.

Mohagheghi, Parastoo and Casper Lassenius (2021). 'Organizational Implications of Agile Adoption: A Case Study from the Public Sector'. In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, pp. 1444–1454. ISBN: 9781450385626. DOI: 10.1145/3468264.3473937. URL: https://doi.org/10.1145/3468264.3473937.

Myers, Michael and Michael Newman (May 2007). 'The Qualitative Interview in IS Research: Examining the Craft'. In: *Information and Organization* 17, pp. 2–26. DOI: 10.1016/j.infoandorg.2006.11.001.

Nikitina, Natalja, Mira Kajko-Mattsson and Magnus Strale (May 2012). 'From Scrum to Scrumban: a case study of a process transition'. In: *2012 International Conference on Software and System Process, ICSSP 2012 - Proceedings*. DOI: 10.1109/ICSSP.2012.6225959.

Oates, Briony J (2006). *Researching Information Systems and Computing*. Sage Publications Ltd. ISBN: 1412902231.

Ohno, Taiichi (1988). *Toyota Production System: Beyond Large-Scale Production*. Portland, OR: Productivity. ISBN: 0-915299-14-3.

Olson, Margrethe H (Mar. 1983). 'Remote Office Work: Changing Work Patterns in Space and Time'. In: *Commun. ACM* 26.3, pp. 182–187. ISSN: 0001-0782. DOI: 10.1145/358061.358068. URL: https://doi.org/10.1145/358061.358068.

Petersen, Kai, Claes Wohlin and Dejan Baca (2009). 'The Waterfall Model in Large-Scale Development'. In: *Product-Focused Software Process Improvement*. Ed. by Frank Bomarius et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 386–400. ISBN: 978-3-642-02152-7.

Phillips, Stephen (Sept. 2020). 'Working through the pandemic: Accelerating the transition to remote working'. In: *Business Information Review* 37.3, pp. 129–

134. ISSN: 0266-3821. DOI: 10.1177/0266382120953087. URL: https://doi.org/10.1177/0266382120953087.

Poppendieck, Mary and Tom Poppendieck (2003). *Lean Software Development: An Agile Toolkit*. USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0321150783.

Radford, Alec et al. (2022). *Robust Speech Recognition via Large-Scale Weak Supervision*.

Runeson, Per and Martin Höst (2009). 'Guidelines for conducting and reporting case study research in software engineering'. In: *Empirical Software Engineering* 14.2, pp. 131–164. ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8. URL: https://doi.org/10.1007/s10664-008-9102-8.

Salas, Eduardo, Dana Sims and Shawn Burke (Feb. 2005). 'Is there a "Big Five" in Teamwork?' In: *Small Group Research* 36, pp. 555–599. DOI: 10.1177/1046496405277134.

Schwaber, Ken (2004). *Agile Project Management With Scrum*. USA: Microsoft Press. ISBN: 073561993X.

Schwaber, Ken and Jeff Sutherland (2020). *The Scrum Guide*. URL: https://scrumguides.org/scrum-guide.html.

*Scrum of scrums - guide to agile scaling frameworks* (May 2016). URL: https://www.agilest.org/scaled-agile/scrum-of-scrums/.

Sharma, Shruti and Nitasha Hasteer (2016). 'A comprehensive study on state of Scrum development'. In: *2016 International Conference on Computing, Communication and Automation (ICCCA)*, pp. 867–872. DOI: 10.1109/CCAA.2016.7813837.

Sharp, Helen and Hugh Robinson (2010). 'Three 'C's of Agile Practice: Collaboration, Co-ordination and Communication'. In: *Agile Software Development: Current Research and Future Directions*. Ed. by Torgeir Dingsøyr, Tore Dybå and Nils Brede Moe. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 61–85. ISBN: 978-3-642-12575-1. DOI: 10.1007/978-3-642-12575-1{_}4. URL: https://doi.org/10.1007/978-3-642-12575-1_4.

Shore, James and Shane Warden (Nov. 2007). *The Art of Agile Development*. ISBN: 978-0-596-52767-9.

Skelton, Matthew and Manuel Pais (Sept. 2019). *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. IT Revolution Press.

Stray, Viktoria, Nils Brede Moe and Rashina Hoda (2018). 'Autonomous Agile Teams: Challenges and Future Directions for Research'. In: *Proceedings of the 19th International Conference on Agile Software Development: Companion*. XP '18. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450364225. DOI: 10.1145/3234152.3234182. URL: https://doi.org/10.1145/3234152.3234182.

Strode, Diane, Torgeir Dingsøyr and Yngve Lindsjørn (Feb. 2022). 'A teamwork effectiveness model for agile software development'. In: *Empirical Software Engineering* 27. DOI: 10.1007/s10664-021-10115-0.

Strode, Diane and Sid Huff (Jan. 2012). 'A Taxonomy of Dependencies in Agile Software Development'. In: *ACIS 2012 : Proceedings of the 23rd Australasian Conference on Information Systems*.

Strode, Diane E et al. (2012). 'Coordination in co-located agile software development projects'. In: *Journal of Systems and Software* 85.6, pp. 1222–1238. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2012.02.017. URL: https://www.sciencedirect.com/science/article/pii/S0164121212000465.

Sugimori, Y et al. (Jan. 1977). 'Toyota production system and Kanban system Materialization of just-in-time and respect-for-human system'. In: *International Journal of Production Research* 15.6, pp. 553–564. ISSN: 0020-7543. DOI: 10.1080/00207547708943149. URL: https://doi.org/10.1080/00207547708943149.

Takeuchi, Hirotaka and Ikujiro Nonaka (1986). 'The New New Product Development Game'. In: *Harvard Business Review*. URL: http://apln-richmond.pbwiki.com/f/New%20New%20Prod%20Devel%20Game.pdf.

Thompson, James D (1967). *Organizations in action: Social science bases of administrative theory.* New York, NY, US: McGraw-Hill, pp. 192, xi, 192–xi.

Ven, Andrew, Andre Delbecq and Jr Koenig (Dec. 1976). 'Determinants of Coordination Modes Within Organizations'. In: *American Sociological Review* 41. DOI: 10.2307/2094477.

Williams, L and A Cockburn (June 2003). 'Agile software development: It's about feedback and change'. In: *COMPUTER* 36.6, pp. 39–43. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1204373.

Woldseth, Tommy (Dec. 2022). *Coordination challenges in large-scale agile software development projects: A literature review*. Tech. rep. (Unpublished: TDT4501

- Computer Science, Specialization Project at NTNU, supervised by: Torgeir Dingsøyr).

Yin, Robert K. (2018). *Case Study Research and Applications - Design and Methods*. 6th ed. Los Angeles: SAGE Publications Inc.

# Appendix

## A Interview guide

The following interview guide was used for all the interviews of both cases. A Norwegian version was used for the majority of the interviews, while the following English-translated version was used in the interviews with non-Norwegian speakers.

### A.1 Intro

- **About me:** I study Computer Science at NTNU, writing my master's thesis this spring.
- **About the thesis:** The topic is "teamwork effectiveness" in the context of large agile projects. I will attempt to discover how teams cooperate in these projects and what promotes effective teamwork.
- **Motivation for this project:** In large projects there are many developers and stakeholders that have to collaborate. I believe this increases the importance of working together in an effective way. I also believe that the use of agile methods increases the importance of effective teamwork because less planning is done up-front which could have identified dependencies.
- **What will the data be used for:** The data collected from these interviews will lay the foundation for a case study. The data will be analyzed, and the findings will be compared to existing models.

### A.2 Practical

- **Time:** 45 minutes
- **Audio recording:** Audio recording is okay? The reason for audio recording is to get a more accurate representation of what was said than I could give with notes and my own memory. The recordings will later be transcribed to be analyzed. Personal information will be anonymized, audio recordings will

be deleted when the project is finished. Audio recordings and transcriptions can be sent for approval if you want.

- **Answers:** Answer in the way you want. My goal is to receive personal opinions and experiences.

### A.3 About interviewee

- Can you tell me a bit about what you're working on now?
- What role do you have in the project?
- How long have you worked on this project?

### A.4 About their project

- How many developers are in the project?
- How many teams?
- Do you know how long this project will go on for?
- How is this project structured? Different teams etc?
- Which agile methods are used?
  - Have there been changes over time?
  - Any specific framework?
- Size of the team? Thoughts on that?
- What roles exist on the team? How many of each? Mix of experience-level or even?
- How does that impact the teamwork?

### A.5 Main part

- What do you think about the teamwork within the project?
  - Have there been changes over time?
- How does communication happen within the team and across teams?
  - Digital/physical, one-to-one or groups, thoughts on this?
- Do you have any specific suggestions as to what has improved the teamwork?
- Do you have any specific suggestions for challenges related to teamwork?
- What tools/routines do you use to accomplish effective teamwork?
  - Do these work well?
- Do you have any suggestions as to how you think the teamwork in the project could be improved?
- Do you have any specific routines for evaluating and improving teamwork or coordination?
- Do you have any thoughts on specific processes that are being used in a positive or negative way (meetings, workflow, integration, deployment, agile methods)?
- Do you use any practices from xp?
  - For example pair-programming, tdd, continuous integration
- How do you coordinate work within the team and across teams?
  - Does this work well?
- How are the seating arrangements?
  - Remote/work from home?
- Do you feel that there are a lot of dependencies with other teams?
  - Are these handled well?

- How is the leadership within the team?
  - One leader who makes decisions or several people or whole the team?
- Could you tell me a bit about how feedback is given between team members?
  - Are the members of the team open to giving and receiving feedback?
- Are there things in the project that only one person is able to do, can everyone do everything, or somewhere in between?
- How would you rate the adaptability of the team?
  - Is the team able to respond quickly to changes? For example changes in requirements or resources available.
- How would you rate the ≪team spirit≫ of the team?
  - Do you feel that team members set the goals of the team or project in front of their own goals?
- Do you have shared mental models within the team?
  - Common understanding of things that everyone knows, but which is not written
  - For example understanding of goals, tasks, processes, routines, etc.
  - Is it known who knows what in the team?
- How is the trust between team members in the team?
  - Can you trust that everyone is able to do their tasks?
  - Or that people ask for help when they need it?
- How would you rate the communication within the team?
  - How do you communicate within the team?
  - Is it easy to get the information you need, when you need it?
- How would you rate the productivity of the project as a whole (is the progression as planned)?

  - How is this connected to the teamwork?

- How do you accomplish knowledge sharing?
  - Does that work well?
- Any specific measures to ensure learning?
- Do you know what the other teams do at all times?
- Would it be easy to come into the project as a new developer, or would it take time to get to know the project?
  - Is the code well documented?

### A.6 Conclusion

- Anything you want to add on the topic that we have not discussed?
- **Next steps:** The data will be transcribed. You can ask for a copy of the data by e-mail (to comply with data collection rules). May take some time to transcribe. Will hold a presentation about the status of the project for the company at the end of April/start of March.
- Any questions?
- Thank you for participating in this interview.

## B Theme codes from NVivo

Below follows two hierarchical overviews of codes from the qualitative data analysis tool NVivo which was used during the analysis phase. The first overview displays the codes that were generated in the coding iteration, while the second overview shows the final five themes that were used in Section 4, along with their sub-codes.

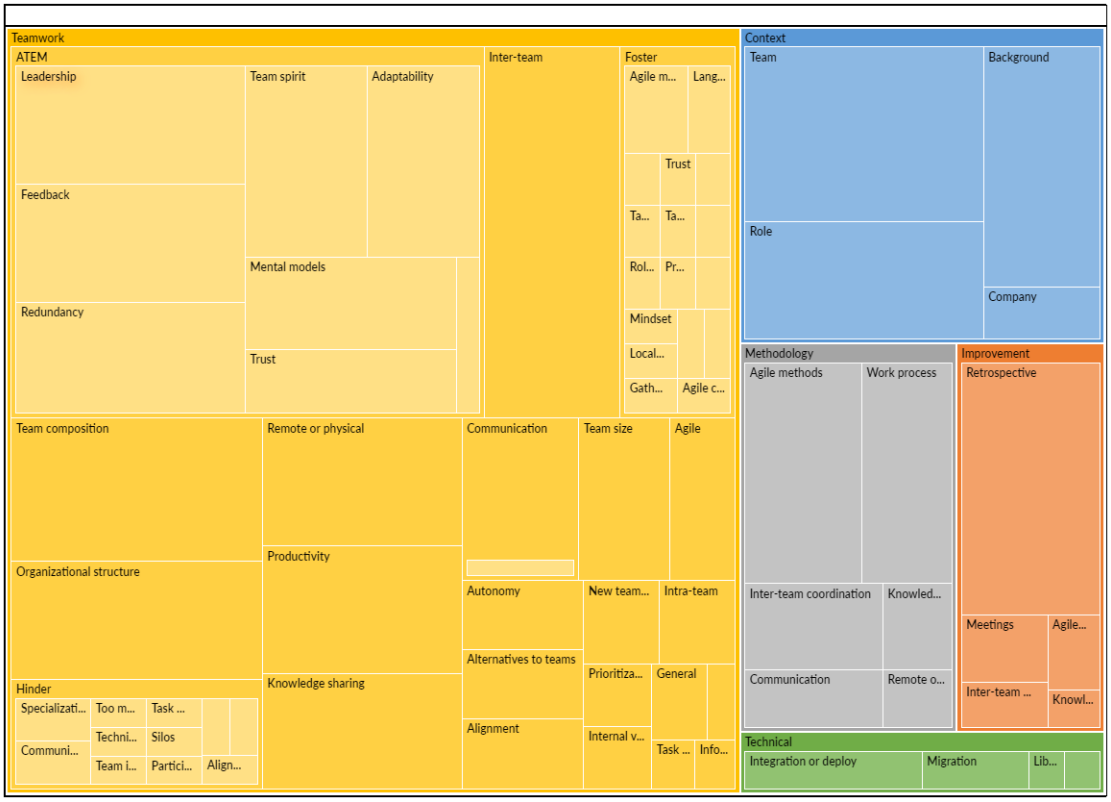



Figure 17: Initial theme codes

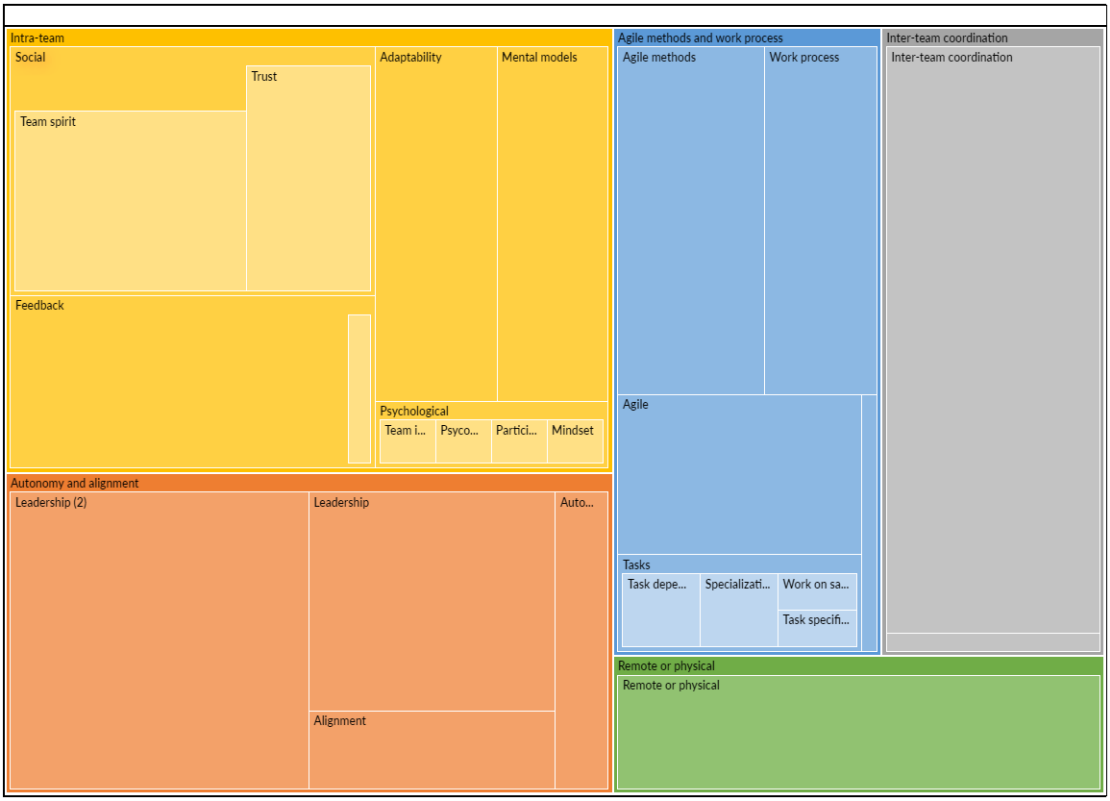
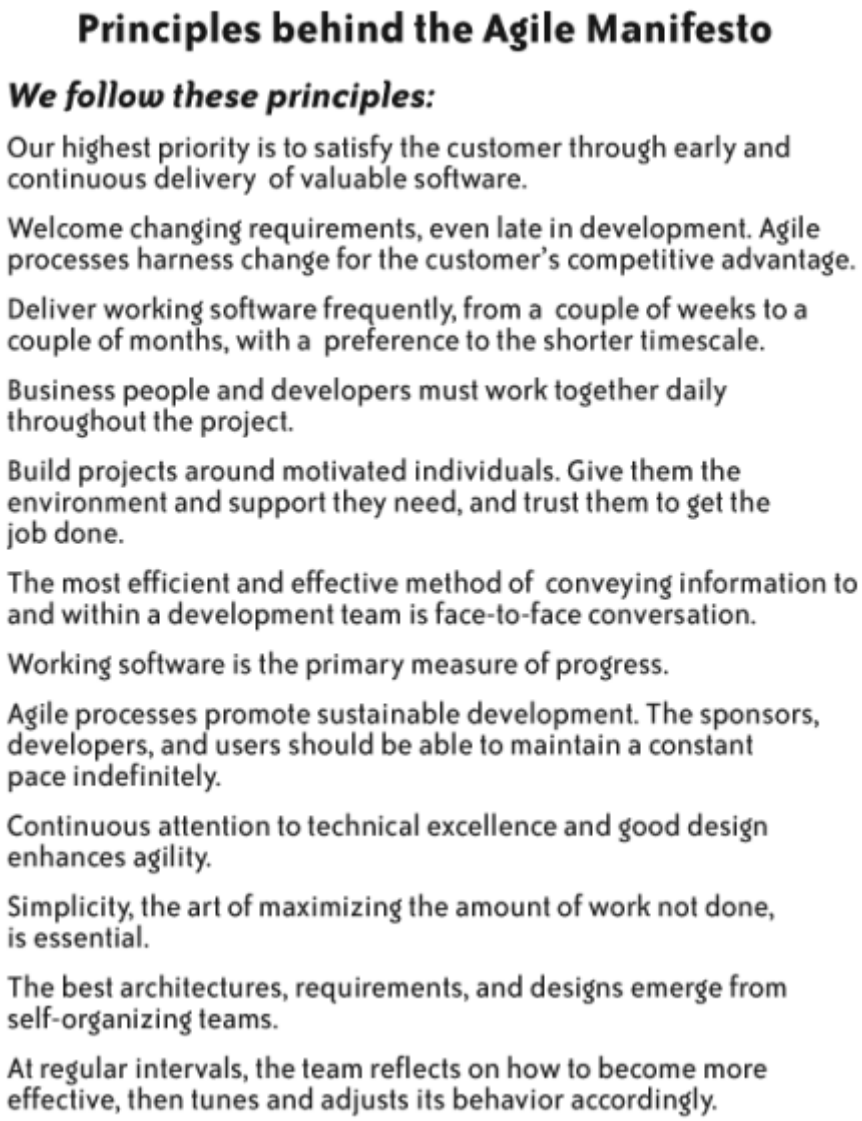



Figure 18: Final five themes

# C Agile principles



## Principles behind the Agile Manifesto

***We follow these principles:***

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

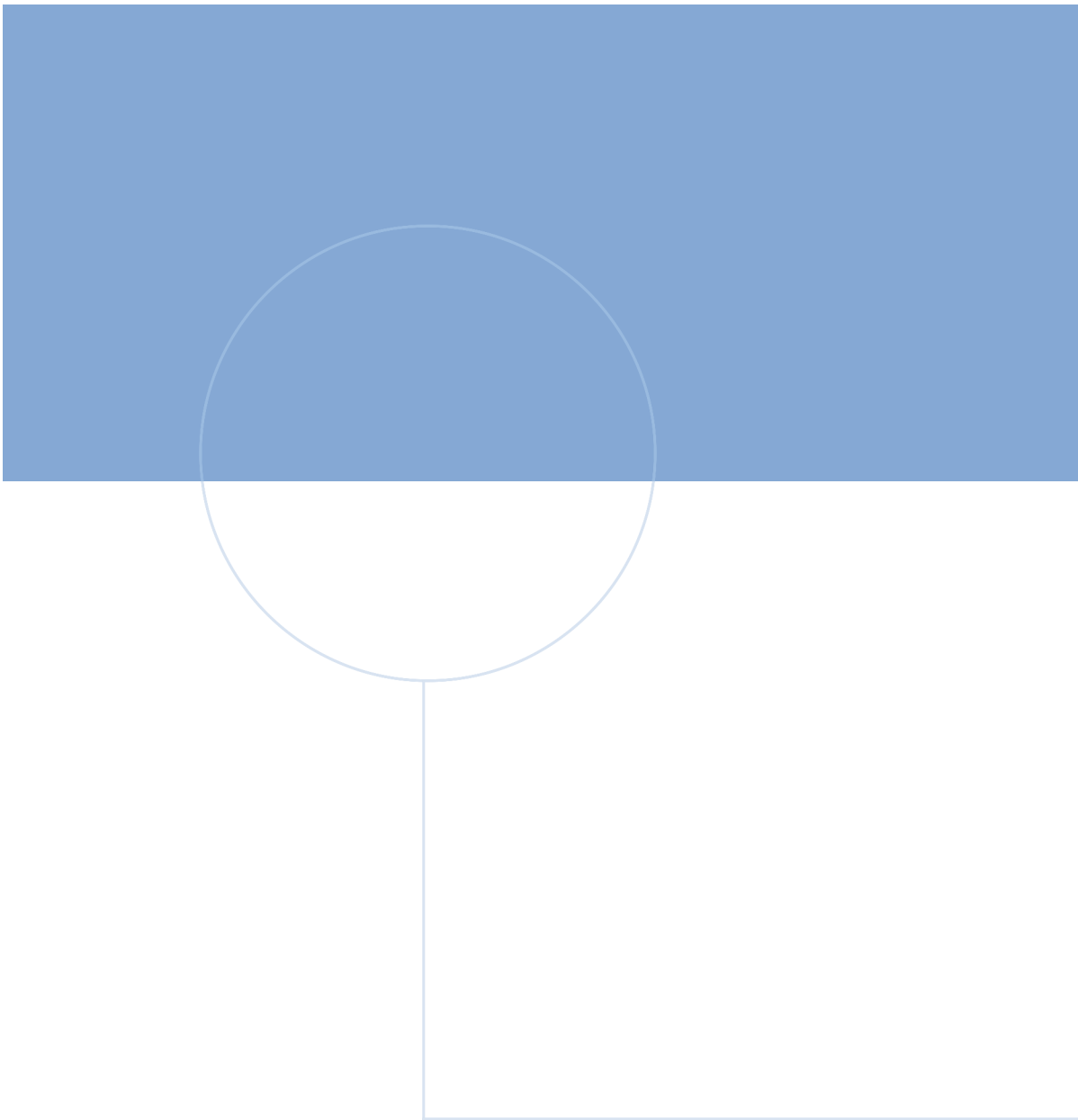
Simplicity, the art of maximizing the amount of work not done, is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Figure 19: Principles of the agile manifesto

Source: (Shore and Warden 2007)



NTNU

Norwegian University of
Science and Technology