Master's thesis

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

Camilla Tøftum Ranner

Large-scale Agile Software Development

An Exploratory Case Study of Changes In Agile Practices and Its Effects on Inter-team Coordination

Master's thesis in Informatics

Supervisor: Torgeir Dingsøyr, IDI

June 2020

NTNU
Kunnskap for en bedre verden

Camilla Tøftum Ranner

Large-scale Agile Software Development

An Exploratory Case Study of Changes In Agile Practices and Its Effects on Inter-team Coordination

Master's thesis in Informatics
Supervisor: Torgeir Dingsøyr, IDI
June 2020

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

Abstract

Agile methodologies were initially created for smaller single-team projects. However, in later years, agile methodologies have also been applied to larger software development projects. This has caused large-scale agile software development projects to face multiple challenges, with inter-team coordination as one of the most prominent challenges. As large-scale agile development projects are becoming more common, and there is found to be a lack of research related to inter-team coordination within this field, this thesis aims to explore this topic further. The stated research question for this thesis is "How is inter-team coordination affected by change of agile methodology in a large-scale agile software development project?". An exploratory case study was chosen as research strategy, where the investigated case was a large-scale agile software development project within a public Scandinavian welfare organization. The project was divided into three phases, where the working methodology changed throughout the project from a "water-scrum-fall" approach in the first phase to working with autonomous and cross-functional teams, and continuous deliveries in the last phase.

The results from this case study is based upon 37 interviews and documentation from the investigated project. The interviews were conducted by researchers from SINTEF, University of Oslo, and the thesis author, in the period of December 2017 to February 2020. The results were analyzed by using a theoretical framework on coordination modes by Ven et al. (1976), and used to compare how the coordination modes were affected by the change of agile methodology.

The findings showed that within the first phase, group mode of coordination were mostly used, followed by personal mode and then impersonal mode. In the last phase it was found to be a large increase in the use of personal mode, a decrease of scheduled meetings and an increase of unscheduled meetings within group mode, while impersonal mode decreased.

There was found a lack of similar research that could substantiate the findings from the last project phase in this case. It is recommended that more research is conducted on inter-team coordination in projects with autonomous- and cross-functional teams, and continuous deliveries.

Sammendrag

Smidige metodikker ble opprinnelig laget for mindre IT-prosjekter bestående av ett team. I de senere årene har smidige metodikker også blitt brukt på prosjekter i stor skala. Dette har forårsaket store smidige prosjekter til å møte på flere utfordringer, hvor koordinering mellom team er en av de største utfordringene. Da storskala smidige IT-prosjekter blir stadig mer vanlig, og det er funnet manglende forskning på koordinering mellom team i en slik kontekst, vil denne masteroppgaven se nærmere på dette temaet. Forskningsspørsmålet for denne masteroppgaven er «Hvordan er koordinering mellom team påvirket av endring i smidig metodikk i et storskala IT-prosjekt?». En eksplorativ casestudie ble valgt som forskningsstrategi, hvor det valgte caset var et storskala smidig IT-prosjekt i en offentlig skandinavisk velferdsorganisasjon. Prosjektet ble delt opp i tre faser, hvor arbeidsmetodikken endret seg gjennom prosjektet, fra en «water-scrum-fall» metodikk i den første fasen, til å jobbe med autonome og kryssfunksjojelle team, med kontinuerlige leveranser i den siste fasen. Resultatene fra denne casestudien er basert på 37 intervjuer og dokumentasjon fra prosjektet. Intervjuene ble gjennomført av forskere fra SINTEF, Universitetet i Oslo, og forfatteren av denne masteroppgaven, i perioden desember 2017 til februar 2020. Resultatene ble analysert ved å bruke en teoretisk modell rundt koordineringsmoduser av Ven et al. (1976), og ble brukt til å sammenligne hvordan kvordineringsmodusene ble påvirket av endring av smidig metodikk. Resultatene viste at i den første fasen, var gruppemodus mest brukt, fulgt av personlig modus, og deretter upersonlig modus. I den siste fasen var det en stor økning i bruken av personlig modus, en nedgang i bruk av planlagte møter og en økning i spontane møter innenfor gruppemodus, og upersonlig modus hadde en nedgang i bruk.Det ble funnet manglende forskning som kan støtte opp om funnene i den siste prosjektfasen av caset. Det anbefales å gjennomføre mer forskning på koordinering mellom team i IT-prosjekter med autonome, kryssfunksjonelle team og kontinuerlige leveranser.

Acknowledgements

I would like to thank my supervisor, Torgeir Dingsøyr, for guidance and support throughout the work with this thesis. He has given me valuable advice and feedback on the content and structure of this thesis. Torgeir provided me with data collected by him and fellow researchers in regards to the investigated case. Without having access to this data, it would not have been possible to conduct this case study within the time scope of this thesis. He also invited me to take part in a guest lecture on coordination with Diane Strode at SINTEF in Trondheim, and a seminar on agile management at Storebrand's office in Lysaker. Both events were very educational and motivational for my work with this thesis. Torgeir is an engaged, knowledgeable and professional supervisor, that I would highly recommend to future graduate students from the Department of Computer Science at NTNU.

I would also like to thank my family and friends for help with proofreading, and for support and encouragement throughout this process.

Camilla Tøftum Ranner

Trondheim, 1 June 2020

Contents

List of Figures

List of Tables

1 Introduction

This section provides motivation behind writing this thesis and the stated research question. It also presents an overview of the thesis structure. Finally, this section describes whom the intended audience for this thesis is.

1.1 Motivation

The digitalization of our society is increasing rapidly. Software development projects are often faced with the need to adjust to changes quickly to keep up with the demands from the customers. Traditional methodologies, such as Waterfall, are based upon thoroughly planning of the whole project process and setting all the requirements, before doing any development. If there is a need for change in requirements midways, the planning and requirement process needs to start over. This cause traditional methodologies to not be suited for projects that are prone to changes. As a response to this, agile methodologies were created. The Agile Manifesto was published in 2001 by a group of software developers (Beck et al., 2001). The manifesto contains core values that creates the foundation for agile methodologies. Based on the manifesto the focus of agile development should be on people rather than processes and documentation, it encourages close collaboration with customers, and one should be able to adapt to changes. Thus, agile methodologies are fit to meet the demands of rapid changes. The most common agile methodology today is Scrum (VersionOne, 2019). Agile methodologies such as Scrum were developed with one team in mind, but recent years there has been an increase in the use of agile methodologies for large-scale agile software development projects. Large-scale scale is defined by two or more teams (Dingsøyr et al., 2014). There are variations of agile methodologies created for large-scale development projects, where Scaled Agile Framework and Scrum of Scrums are the most popular ones (VersionOne, 2019). Scaling agile has however been debated as agile principles was initially created for single teams, and there are multiple challenges related to large-scale agile projects. Coordination in multi-team environments is found to be one of the more prominent challenges (Dikert et al., 2016). The academic field of large-scale agile development is quite a new field, and there has not been conducted much research on coordination challenges within this topic yet. There has been some research conducted on the area of coordination challenges in regards to agile development projects (Strode, 2016; Stray et al., 2019), but these do not study multi-team projects. As large-scale agile development becomes more common, there is an increased need for understanding the challenges such projects are faced with. Based on this, the chosen topic for this thesis will be inter-team coordination for large-scale agile software development projects.

1.2 Research Question

This thesis investigates a case where a large-scale agile software development project has gone from a "water-scrum-fall" methodology to working with autonomous, cross-functional teams, with continuous deliveries. By investigating this case, one can compare the differences in inter-team coordination, and see how coordination has been affected by this change of methodology. Thus, the research question for this thesis will be:

How is inter-team coordination affected by change of agile methodology in a large-scale agile software development project?

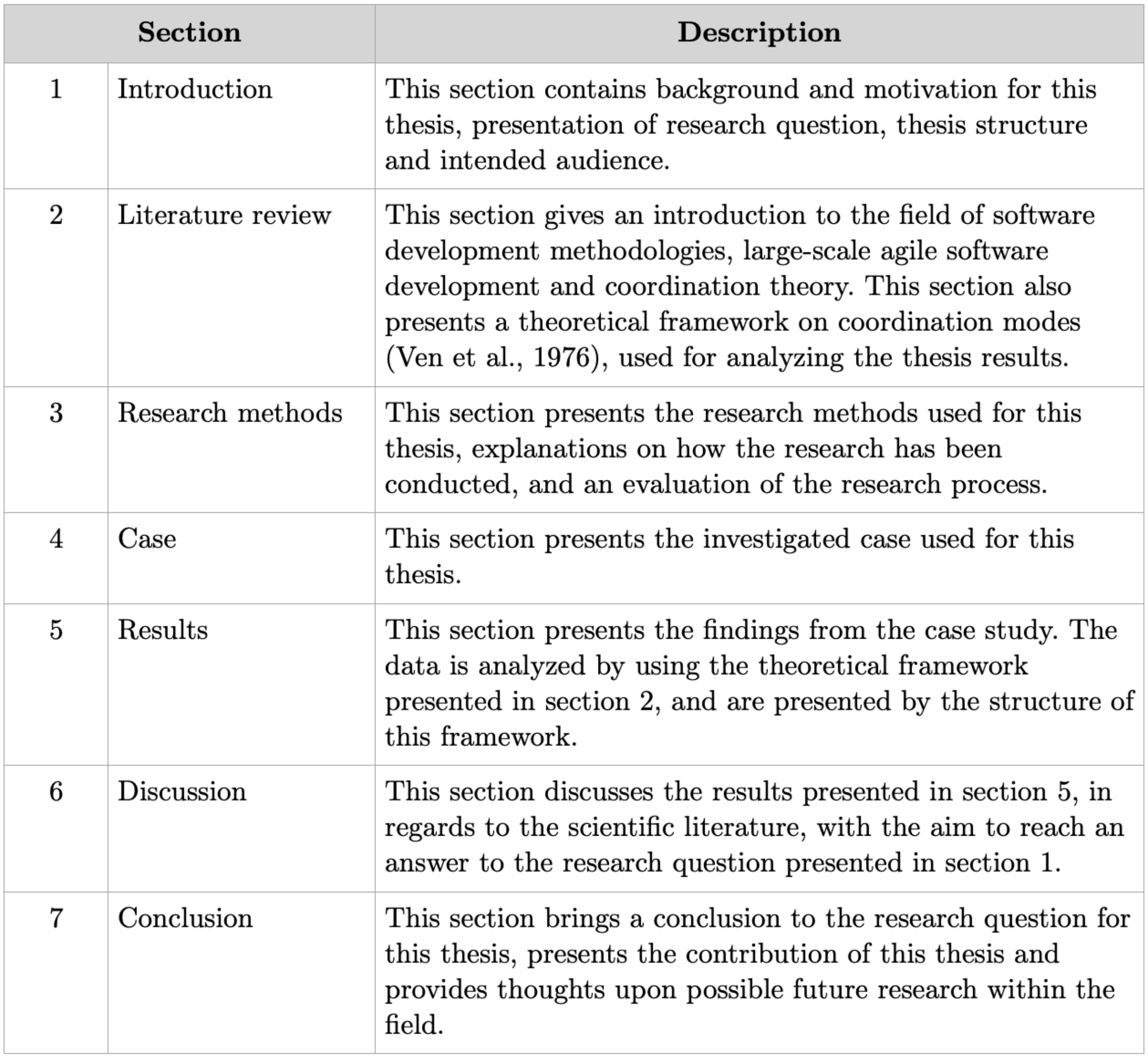
1.3 Target Audience

This thesis explores inter-team coordination in regards to large-scale agile software development, and could be found interesting for practitioners within agile software development. As the investigated project in this thesis transition from one agile methodology to another, the findings of this thesis can be insightful for project managers that consider to go through a similar transition for their project. Section 7.4 provides suggestions for future research based on the contribution of this thesis, which can be interesting for researchers and students within this field of topic.

1.4 Thesis Structure

Table 1 gives an overview of the thesis structure, with descriptions of each chapter.

Table 1: Overview of thesis structure



Section		Description
1	Introduction	This section contains background and motivation for this thesis, presentation of research question, thesis structure and intended audience.
2	Literature review	This section gives an introduction to the field of software development methodologies, large-scale agile software development and coordination theory. This section also presents a theoretical framework on coordination modes (Ven et al., 1976), used for analyzing the thesis results.
3	Research methods	This section presents the research methods used for this thesis, explanations on how the research has been conducted, and an evaluation of the research process.
4	Case	This section presents the investigated case used for this thesis.
5	Results	This section presents the findings from the case study. The data is analyzed by using the theoretical framework presented in section 2, and are presented by the structure of this framework.
6	Discussion	This section discusses the results presented in section 5, in regards to the scientific literature, with the aim to reach an answer to the research question presented in section 1.
7	Conclusion	This section brings a conclusion to the research question for this thesis, presents the contribution of this thesis and provides thoughts upon possible future research within the field.

2 Theory

This section presents theory on different software development methodologies. First the concept of software development methodologies will be introduced. Then, examples on popular methodologies are given to provide insight on how software development is practiced. Furthermore, this section will look into the area of large-scale projects in regards to agile development, with associated challenges and examples on frameworks. Finally, coordination theory will be covered and related to large-scale agile software development.

2.1 Software Development Methodologies

A software development methodology consists of a set of processes and activities that will contribute to the creation of a software product (Sommerville, 2011, p. 28). Sommerville describes four activities that must be included when developing a software product:

1. Software specification

   Functionality and requirements must be specified.

2. Software design and implementation

   Functionality must be implemented to fulfill the requirements.

3. Software validation

   Testing must be done to ensure it satisfies the requirements.

4. Software evolution

   Software might need changes and further development to meet the customer needs.

How these activities are structured and performed will depend on what methodology is being used. Using a software development methodology is important as it helps planning and structuring the project and its activities in such a way that will increase effectiveness of the team and reduce costs. Software development processes can mainly be divided into two types of approaches: plan-driven, also known as traditional development, and agile development.

A hybrid approach is also common, which is a combination of agile methodologies and/or plan-driven methodologies (Vijayasarathy and Butler, 2015).

2.2 Plan-driven Software Development

Plan-driven software development refers to development methodologies where all process activities are planned in advance (Sommerville, 2011, p. 29).

2.2.1 Waterfall

The waterfall method was one of the first traditional ways of approaching a software development process. The stages of the waterfall model was introduced by Royce in 1970 (Royce, 1970), but the term "waterfall" was first used by T. E. Bell and T. A. Thayer in 1976 (Bell and Thayer, 1976).

Figure 1 shows an illustration of the waterfall model. The model consists of five stages, that are performed in a linear-sequentially order:

1. Defining requirements

   Understanding and documenting the needs in terms of the design and functionality of the product.

2. System design

   Specifying the technical software and hardware design.

3. Implementation

   Coding the system, based on requirements and plans from the previous steps.

4. Testing

   Testing the system to uncover problems and mistakes .

5. Maintenance

   Deploying the system, and keep it maintained and upgraded.

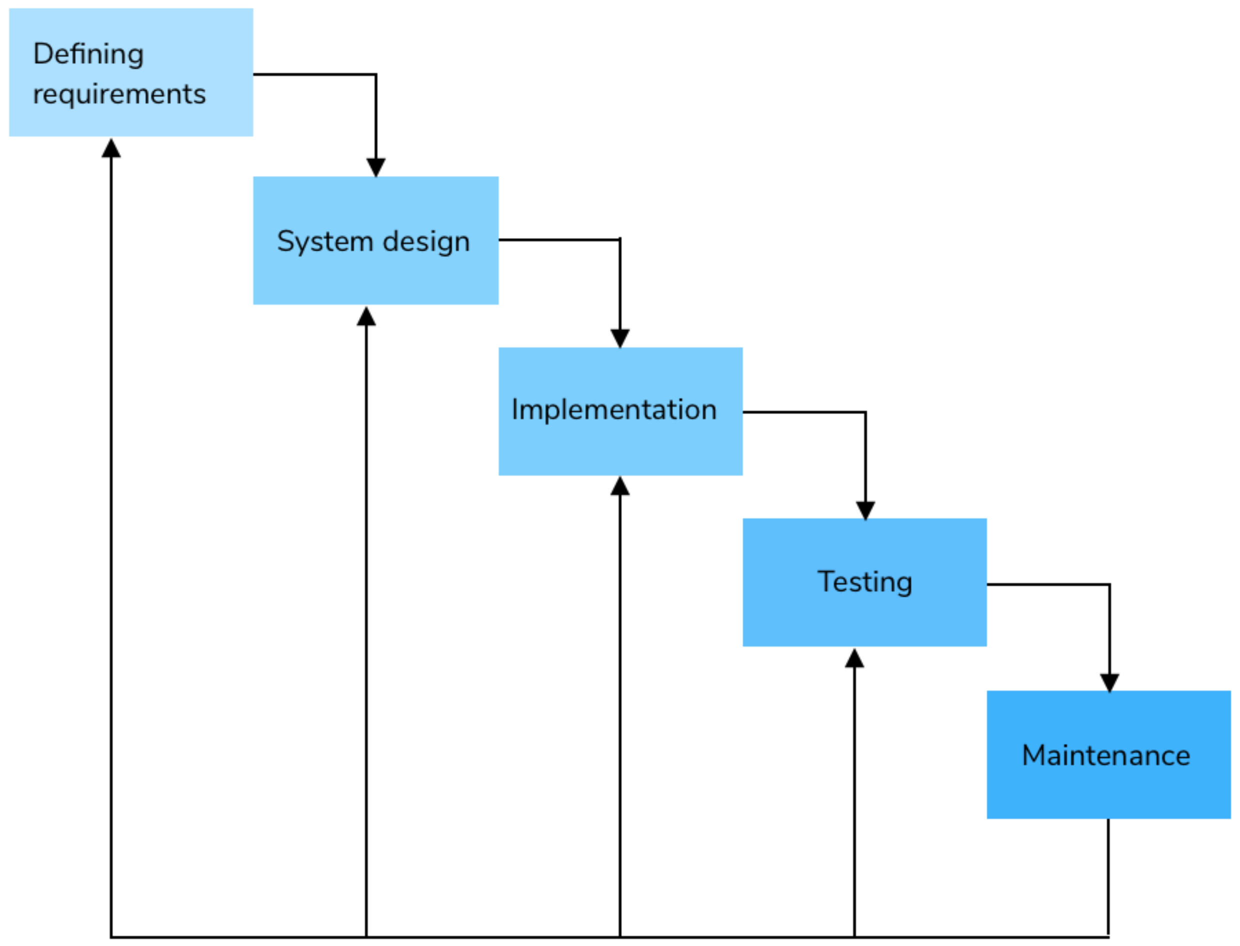


Figure 1: The waterfall model

Each step should be finished before you proceed to the next. If you discover problems during the project or experience failure at the end of the project, you can go back and start the process once more. This was also suggested by Royce (1970), who stated that the cycle should be repeated twice. A strict linear process without repeating any steps would be risky and increase the chances of failure. By going back and repeating the process, one could rule out critical mistakes before delivering the final product to the customer. As goals and requirements are clearly defined early on in the project, one can not simply make changes in the middle of the project or go back to only redo the previous step. One would need to go back to the first step and redo the whole process.

On the other hand, doing the whole process a second time does not guarantee the system will be perfect. In *The Mythical Man Moth* (Brooks, 1975) Brooks states that the first system almost always needs to be redone, but he also introduced the risk of "second system syndrome". The second system syndrome describes how a second version of a system might get over-engineered and include excessive functionality. This is due to increased confidence and ambitions, and the architects may want to improve the first system more than what is necessary. The waterfall model's inflexibility is one of its main weaknesses and this makes it not suitable for projects that are vulnerable to changes. Table 2 shows an overview of the advantages and disadvantages of the waterfall model.

Table 2: Advantages and disadvantages of the waterfall model

Advantages	**Disadvantages**
• Simple to understand due to its clearly defined steps • Suits smaller projects where the requirements are easy to define • Projects are well documented • Heavy planning makes it easier to calculate costs and set deadlines	• Inflexible. Not possible to change requirements in the middle of the project. • Little customer involvement during the project may cause the customer to be less satisfied with the end product. • Higher risk of failure, as problems often are discovered during testing late on in the project. • Risk of encountering "the second system syndrome"

A survey from 2015 shows that 32% of the 153 respondents used waterfall as software development methodology for their project, which indicates that it is still a much used methodology (Vijayasarathy and Butler, 2015).

2.3 Agile Software Development

During the 1980s and 1990s computers became more common, and the introduction of World Wide Web to the public in 1993 made big changes to the economy (J. Highsmith, 2000). This caused an increased demand on software products. Figure 2 shows how the internet usage grew rapidly during the mid 90s .

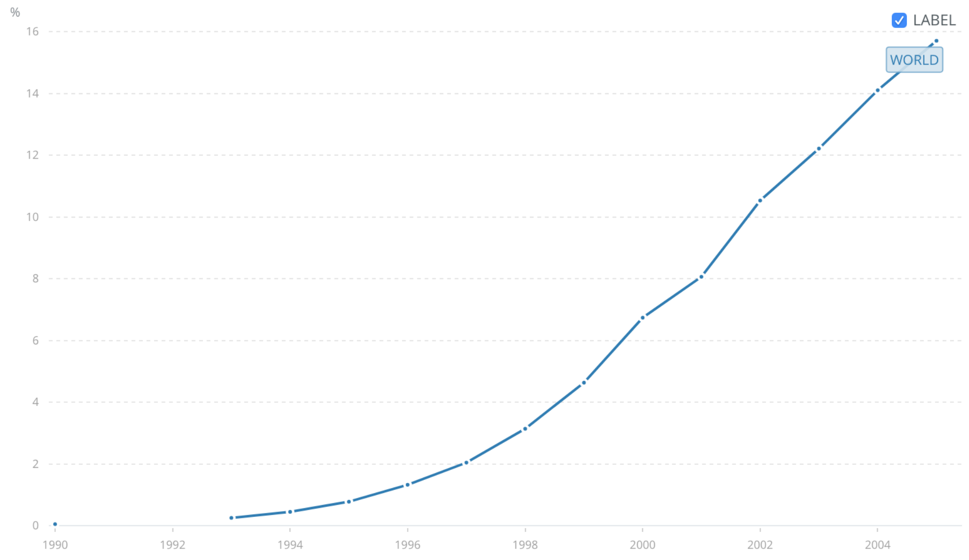


Figure 2: Worldwide Internet usage from 1990 - 2005 International Telecommunication Union and database (2019).

Plan-driven software development was time consuming and inflexible, and by the time the software products were finished, the demands had changed and the projects often either failed or lead to dissatisfied customers. Highsmith stated "In our Information Age economy, a company's ability to set the pace, to create change, lies in its ability to develop software. In a world of constant change, traditional rigorous software development methods are insufficient for success."(Highsmith, 2002, p. 2).

In order to keep up with the changing demands from the customers, more lightweight approaches to software development became common. Kent Beck developed extreme programming (XP) during the late 1990s, and published his book "Extreme Programming Explained: Embrace Change" on the methodology in October 1999 (Beck, 1999). As the title suggests, one should start embracing change, which is in big contrast to the traditional plan-driven development. XP focuses on being highly flexible and making it easier to make changes to requirements by having short development iterations, a lot of customer interaction, and frequent deliveries.

The high failure rates and the struggle with keeping up with the rapid changes when using plan-driven development caused great frustration among developers. As a reaction towards this, Manifesto for Agile Software Development was published in 2001 by a team of software developers, including Kent Beck and Jim Highsmith(Beck et al., 2001). The agile manifesto is a result of the participants experiences and thoughts upon how to approach software development and how to solve the problems with plan-driven methodologies.

The agile manifesto contains four core values of agile software development:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

These values creates the basis for all agile methodologies. Agile development is about being able to adapt to changes by having small iterations and frequent contact with the customer. It also focuses on the people who are involved in the projects, and the collaboration between them.

Findings by Deloitte Center for Government Insights presented by Viechnicki and Kelkar (2017), includes over 3000 respondents from US federal IT projects, and shows how the usage of agile and iterative methods has developed over the years. Figure 3 shows that agile and iterative methods has had an immense growth the last years, and the usage was in 2017 around 80%. Deloitte have combined data from four different sources, but they do not present how these data were extracted, nor how these sources conducted their research. One can therefore not state that these data are completely reliable, but due to the large number of respondents, it might indicate a trend.

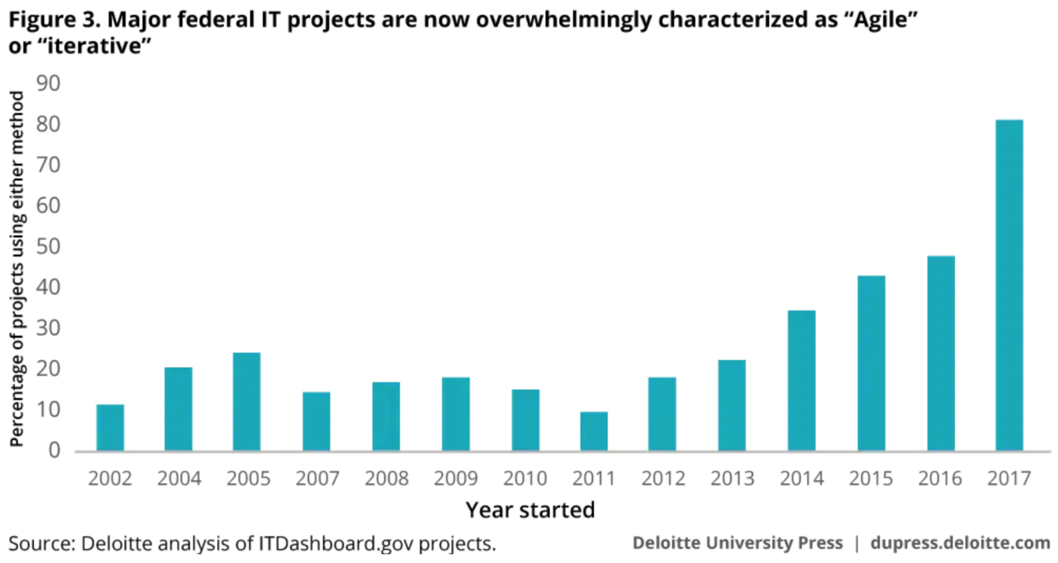


Figure 3: Percentage of projects using iterative or agile methods 2002 - 2017 (Viechnicki and Kelkar, 2017).

2.3.1 Scrum

Scrum is an agile software development framework that has roots back to 1986 when the term was first used (Takeuchi and Nonaka, 1986). It was then described as a new approach to product development with focus on speed and flexibility. The name "Scrum" originates from rugby, where each team forms together and push against the opposing team, trying to gain possession over the ball (Dictionary.cambridge.org, 2019). Just as in rugby, Scrum teams in software development are self-organizing and work together in a fast forward and flexible way to reach their goal.

Jeff Sutherland used Takeuchin and Nonaka's concept of Scrum as a basis when he formulated the early versions of the framework for software development teams in 1993 at Easel Corporation (Sutherland, 2001b). Together with Ken Schwaber, they co-developed Scrum throughout the 1990s. Some years later, Schwaber and Beedle (2001), wrote the first book on Scrum called *Agile Software Development with Scrum*, which contains information on what Scrum is, why it works, and how to practice it. Schwaber and Beedle was also involved in the creation of the Agile Manifesto from the same year.

As of 2019, Jeff Sutherland and Ken Schwaber are still working with Scrum, and together they have written the Scrum Guide, which offers a thorough guide to Scrum. Sutherland and Schwaber defines Scrum as "A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value." (Sutherland and Schwaber, 2017, p.3). Figure 4 shows an illustrated overview of the framework.

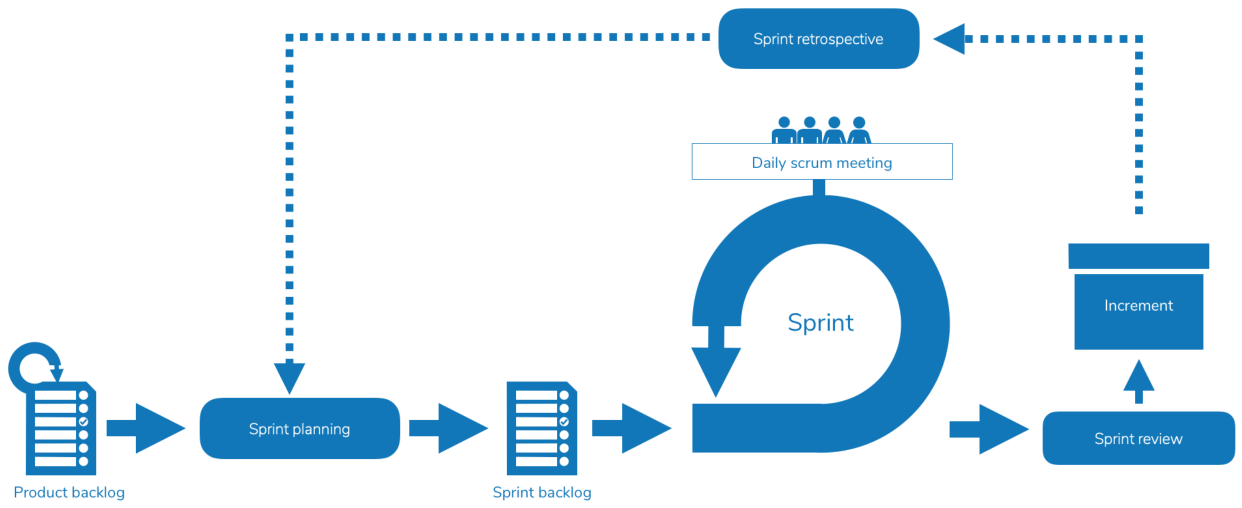


Figure 4: Illustration of the Scrum Framework. Adapted from Sutherland and Schwaber (2017).

Scrum was made for small teams, where each part of the team has a defined role. A Scrum team has a Scrum master, a product owner, and a development team. The Scrum master's job is to make sure the team understand and follows the processes and activities that comes with the Scrum framework. The product owner is responsible for the product backlog and makes sure that the team creates a product of high value that meets the requirements. The development team are responsible for creating a product that meets the requirements. The development team is self-organized and may consist of people with different specializations, depending on what qualifications the team needs to meet the requirements.

Scrum consist of five main events with set time frames. The events are designed to let the team plan and discuss what they are working on, and make room for adaptions. The five main events are:

1. The Sprint

   A sprint is a period of time, set to maximum one month or less, where implementation takes place. The other events will also take place during a sprint.

2. Sprint Planning

   Sprint planning happens at the beginning of a sprint and lasts maximum eight hours. The team will discuss the product backlog and decide on which tasks that should be a part of the sprint. How they will work and how they will achieve their goal for the sprint will also be decided during sprint planning.

3. Daily Scrum

   Daily scrum, often called stand up, is a daily event that lasts maximum 15 minutes. The team members share what they have done the last 24 hours, and discuss what they will work on up until the next meeting.

4. Sprint review

   Sprint review is conducted at the end of each sprint and last maximum four hours. For this meeting, key stakeholders, the product owner and the rest of the Scrum team are present. They discuss what requirements has been met and possible changes that may be done to the product backlog. They also review the product's market value, project budget and timeline. The result of this meeting will work as a basis for the next sprint planning meeting.

5. Sprint retrospective

   Sprint retrospective meeting lasts maximum three hours, and is held after the sprint review. The Scrum team inspect themselves and how the current sprint has been. They discuss what has worked well and what could have been better, in regards to tools, processes and how they work together as a team. The team creates a plan for what can be improved and how they will solve it. The purpose of this is to enhance how they work and avoid doing the same mistakes as earlier during the next sprint.

As seen in Figure 4 there are three artifacts that is an important part of a Scrum process. These are: product backlog, sprint backlog, and increment. The product backlog describes all planned functionality and requirements for the product. The sprint backlog contains requirements and tasks from the product backlog that are planned to be completed during the current sprint. The increment is a result of all tasks and requirements that has been fulfilled during the sprint, and this part of the product must be considered to be finished and usable. All team members must agree on what is considered to be finished. Having transparency, with clear definitions of different events and artifacts is important as it helps the team have common understanding and expectations of the whole process and the decisions being made.

The 13th Annual State of Agile Report(VersionOne, 2019), shows the current state of agile development and is based on a survey with 1319 respondents, where most respondents are located in North-America and Europe. The report shows that 54% percent of the respondents' organizations use Scrum as their agile software development approach. This indicates that Scrum is currently the most popular framework for agile software development. Despite its popularity, a case study indicates that one may also face issues when using Scrum (Cho, 2008). The study mention that even though Scrum is supposed to have frequent contact with the customer, this is not always the case when practiced in real-life. The teams struggle to keep in touch with the customer as often as they would like, because the customers often are too busy to talk with them. Another challenge mentioned in the study is that some people find the framework to be too rigid. The teams sometimes experienced that the required meetings and events was not always beneficial and felt like a waste of time. One should rather have meetings based on the complexity of the project, and when the teams feel it is necessary. An overview of advantages and disadvantages of using Scrum is presented in Table 3.

Table 3: Advantages and disadvantages of Scrum

Advantages	**Disadvantages**
• Working iterative and incremental with frequent customer interaction makes Scrum very flexible and one could easily adapt to changes • Frequent deliveries with working software after each sprint • Focus on transparency and open information flow between team members • Planned events that enhance communication and collaboration among team members	• Not as simple to understand and implement as Waterfall, and requires skilled project managers • Scrum is best suited for small teams. Scaling Scrum can get too complex. • Due to its flexibility it can be difficult to set a final project deadline • Scrum projects are vulnerable to scope creep

2.3.2 Autonomous and cross-functional teams

As stated in the Agile Manifesto, "The best architectures, requirements, and designs emerge from self-organizing teams." (Beck et al., 2001). Agile teams should therefore be self-organized, also known as *autonomous*. This means giving freedom to the teams to decide how they organize and practice their work to reach their goals. Autonomy can increase satisfaction and motivation among team members, and increase productivity (van Mierlo et al., 2006). As the teams are self-organized, project managers can spend less time on management at team level. Thus, there are many advantages with using autonomous teams.

However, it is also found some challenges in regards to autonomous teams within agile software development. Moe et al. (2010) have found multiple challenges related to autonomous teams, both in regards to the surrounding organization as an external factor that affects the team, but also within the team itself. On team-level, "individual commitment", "individual leadership", and "failure to learn" were considered as challenges. Individual commitment is related to how team members may focus on individual goals within their field of specialization, rather than pursuing the team goals. Individual leadership looks at how the teams handle decision making. Within autonomous teams, individuals may take charge and make decisions without informing team members, which could lead to mistrust. Shared leadership was found to be difficult, and it can be challenging to decide who should take part the different decisions. Failure to learn is about the teams ability to progress and improve themselves. Stray et al. (2018a) also present challenges related to autonomous teams. One of the challenges mentioned was that autonomous teams may struggle with too many dependencies to others.

Within the practice of Scrum it is also encouraged to use cross-functional teams. This means including roles across different fields of expertise from the organization. By doing this the teams have the knowledge they need to perform tasks without the need to rely on people outside the team. By using cross-functional teams, the team members will be able to see the project from different perspectives, gain an understanding of each others knowledge, and also be a part of tasks outside their field of specialization (Hoda et al., 2012). Relationships that contains shared knowledge, shared goals and mutual respect is said to be important for coordination. When such relationships develop within the teams, it strengthens the team and creates a foundation for good collaboration (Gittell, 2006).

Combining the use of autonomous and cross-functional teams may therefore reduce the need for coordination with people outside the teams, as all necessary domain knowledge can be found within the teams.

2.4 Large-scale Agile Development

Agile development were initially used by smaller software development projects consisting of single teams. The Scrum Guide recommends the development team size to be no less than three members, and maximum nine members. Having more than nine members would be complex and require too much coordination (Sutherland and Schwaber, 2017, p.7). As early as 2003 it was not recommended to scale agile projects (Reifer et al., 2003), and Dybå and Dingsøyr (2009) mention that introducing agile methods to large and complex projects can be difficult.

Despite multiple sources discourage scaling agile projects, agile methods has become more common for larger and more complex projects. A reason for this may be the successful results that agile development has provided for smaller teams, which have inspired further use of agile approaches (Dingsøyr and Moe, 2014). Another contributing factor is the increasing digitalization in our society and a rapidly growing market for digital technology (Dingsøyr et al., 2019). This creates a need for both small and large software development projects to be flexible and able to adapt to changes at a fast pace.

2.4.1 Definition of Large-scale Agile Projects

There is no official definition for what a large software development project is, but it is suggested to be measured by the number of people involved in the project, number of lines of code, and the duration of the project (Rolland et al., 2016). In the article *What is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development* (Dingsøyr et al., 2014) the authors identify three types of projects, based on the numbers of teams involved and their coordination approaches.

Table 4: A taxonomy of scale of agile software development projects (Dingsøyr et al., 2014, p. 4).

Level	**Number of teams**	**Coordination approaches**
Small	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.

The taxonomy is presented in Table 4, and from this taxonomy one can define a large project as a project with two or more teams. Thus, a large-scale agile software development project can be defined as a software development project involving two or more teams, that use agile methods for their project. This will be used as the definition for a large-scale agile software development project throughout this thesis.

2.4.2 Scaled Agile Framework (SAFe)

Scaled Agile Framework (SAFe) was first introduced in 2011 as "Agile Enterprise Big Picture" (Leffingwell, 2011). It was a framework that described how enterprises can implement agile and lean practices on three different levels: portfolio level, program level and team level. Since 2011 the framework has developed and it has been released multiple versions. As of November 2019, the latest version was released in October 2018 and is known as "Scaled Agile Framework (SAFe) 4.6". This version includes principles from Lean, DevOps and Agile practices.

SAFe has four core values they describe as the key to its effectiveness: alignment, built-in quality, transparency and program execution (© Scaled Agile, Inc., 2019a).

Values and principles, along with a SAFe Program Consultant, an implementation roadmap, and a Lean-Agile- mindset and leadership, creates the foundation for the SAFe framework. Figure 5 shows an illustration of full SAFe 4.6.

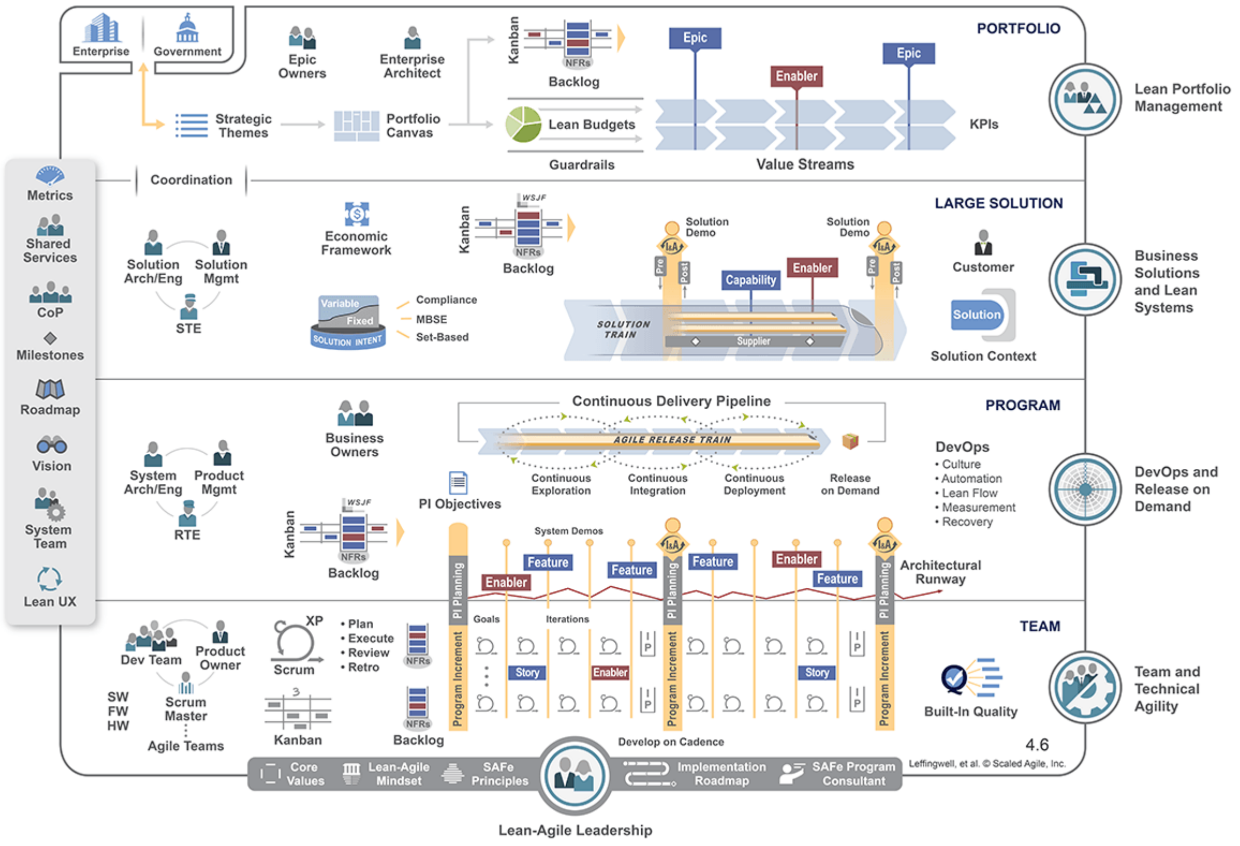


Figure 5: Illustration of full SAFe 4.6 (© Scaled Agile, Inc., 2019b)

Full SAFe includes all levels of SAFe: team level, program level, large solution level and portfolio level. Each level consists of different processes, artifacts and roles. It is structured in a hierarchical system. A simplified view of the structure is illustrated in Figure 6.

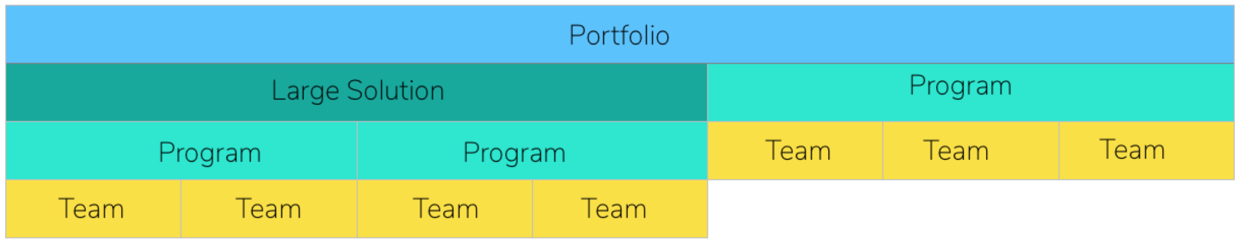


Figure 6: Illustration of the SAFe structure

Team level:

On the bottom you find team level. Team level consists of multiple smaller teams that practice agile methods such as Scrum, XP or Kanban.

Program level:

The program level manage all teams on team level. Multiple teams combined forms a program. This level has different roles that helps facilitate a program: release train engineer, system architect for the entire program, product management, business owners. This is also called Essential SAFe, and is the simplest configuration of SAFe.

Large Solution level:

Large Solution consists of multiple programs combined. To manage these programs you need: Solution Train Engineer, Solution Architect, Solution Management, Customer.

Portfolio level:

The Portfolio level consists of either a Program or a Large Solution. The roles for this level are: EPIC owners, Enterprise Architect, Lean Portfolio Management.

From the 13th Annual State of Agile Report(VersionOne, 2019), 30% of the respondents mentioned they have been using SAFe. This makes SAFe currently the most used method for scaling agile.

2.4.3 Scrum of Scrums

Scrum of Scrums(SoS) was originally defined by Jeff Sutherland when introduced to IDX Systems, a large healthcare software company, in 1996 (Sutherland, 2001a). Jeff Sutherland and Scrum Inc. provides a guide, *The Scrum At Scale® Guide*, which describes SoS and its associated roles and events (Sutherland and Scrum Inc., 2019). SoS is an approach to use Scrum at scale. When more than one Scrum team is needed for a project, one can use SoS for collaboration and to coordinate work among the teams. When using SoS, a representative from each team meet up, and together they form a SoS team. For even larger projects, one can also expand the Scrum of Scrums to involve more than one Scrum of Scrums. This will create a Scrum of Scrum of Scrums. The SoS structure is illustrated in Figure 7.

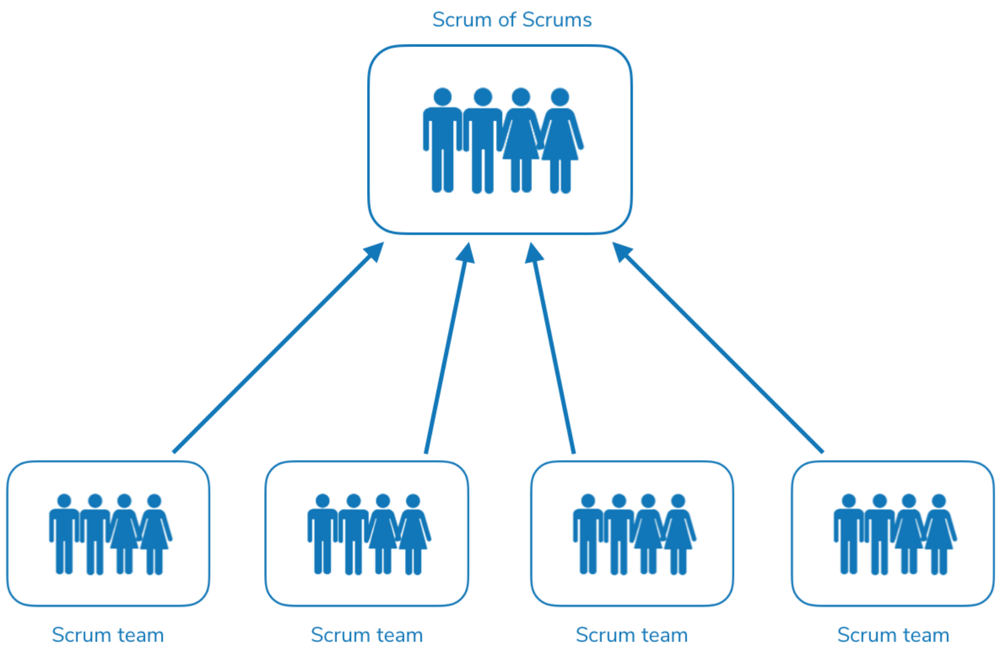


Figure 7: Illustration of the Scrum of Scrums structure

The SoS team meet once a day and have their own stand-up meeting, after each Scrum team have finished their daily stand-up meeting. This way the teams can be up to date on each others work.

The SoS team have multiple roles associated with the team to distribute the different responsibilities. Each SoS team has a SoS Master, Product Owner Team, and a Chief Product Owner. The SoS Master is responsible for making sure that each Scrum team has a progress and deliver a working increment at the end of each sprint. The Product Owner Team manages a backlog that aligns and prioritizes all the tasks from each teams' backlog. They also define when an increment is considered done, and resolve dependencies among the teams. When having a Scrum of Scrum of Scrums structure, there are also multiple Product Owner Teams. The Chief Product Owner is responsible to manage the Product Owner Teams. The Chief Product Owner have the same tasks as a Product Owner, but is responsible for the project as a whole. If there is only used a SoS structure, there is no need for a Chief Product Owner, as the Product Owner can fulfill this role.

When having multiple Scrum of Scrum of Scrums in an organization, an Executive Action Team (EAT) is responsible for coordinating all teams and making sure they implement agile values and Scrum correctly. The EAT is also considered a Scrum team, and has a Scrum Master, Product Owner and a backlog. How an EAT team with five groups of Scrum of Scrum of Scrums is structured is illustrated in Figure 8.

The 13th Annual State of Agile Report shows that Scrum of Scrums is being used by 16% of the respondents VersionOne (2019). Scrum of Scrums is currently being the second most used approach for scaling agile, after SAFe.

2.4.4 Challenges

Agile methods were initially meant for smaller projects, but they do not take into account the challenges that one may face when trying to scale agile methods. A recent study by Jørgensen (2019) looked into the performance of large-scale agile projects, compared to smaller agile projects and non-agile projects. Jørgensen found that only 7% of the large agile projects were categorized as successful, and that the success factor decreased with the increase of project size. This indicates that there are challenges that may be difficult to solve when working with large-scale agile software development projects.

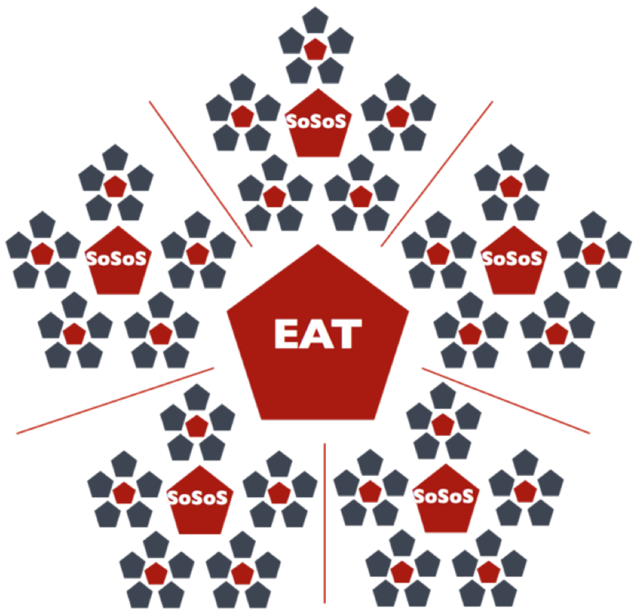


Figure 8: Illustration of EAT structured with five Scrum of Scrum of Scrums (Sutherland and Scrum Inc., 2019).

As software development projects often involves cooperating with stakeholders outside the development team, it is interesting to look at how large-scale agile methods are being approached by organizations and not only by the development teams alone. A study conducted by Dikert et al. (2016) found multiple challenges in regards to large-scale agile transformations in organizations. This study include companies that only focus on software development, but also parts of non-software development organizations that develop software. The most prominent challenges, which was mentioned by more than 30% of the investigated projects, are:

- Agile difficult to implement
- Integrating non-development functions
- Change Resistance
- Requirements engineering challenges
- Hierachical management and organizational boundaries
- Lack of investment

- Coordination challenges in multi-team environment

In the article *Implementing Large-Scale Agile Frameworks: Challenges and Recommendations* by Conboy and Carroll (2019), they identify nine challenges connected to large-scale agile transformations in organizations:

- Defining concepts and terms
- Comparing and contrasting frameworks
- Readiness and appetite for change
- Balancing organizational structure and frameworks
- Top-down versus bottom-up implementation
- Overemphasis on 100% framework adherence over value
- Lack of evidence-based use
- Maintaining developer autonomy
- Misalignment between customer processes and frameworks

One of the challenges mentioned by both studies is that agile is difficult to implement, and that it is hard to define concepts and terms. Conboy and Carroll (2019) describe that when frameworks such as SAFe and Spotify are applied, they may work well for basic implementations, but misunderstandings and difficulties are faced when the frameworks are being adapted to fit different contexts. In general there is a lack of guidance, and vague explanations make room for misunderstandings.

Another challenge mentioned by both of the studies is the openness to change. Team members may be skeptical to change if they do not see any visible benefits or if the manager do not set clear goals for its purpose. Managers can be hesitant to change if the agile transformation process starts bottom-up within a development team and the manager is not being included in the process. Agile methods can cause changes to management roles, and if agile methods and the possible outcomes are not understood properly by managers, they may be unsure of how it will affect their role and are not be willing to pursue these changes any further (Dikert et al., 2016). Conboy and Carroll (2019) describe that managers may not be willing to adopt different frameworks to their practices. Adopting frameworks does not necessarily guarantee that it will work well and that changes will happen. Some organizations that tried to introduce different frameworks experienced frustration among the teams, and some teams would not adapt to these changes in their work at all.

From these studies one can see that multiple organizations are working with agile transformations. Many challenges are faced at a management level, and often related to the difficulties in understanding agile approaches and the willingness to adapt to this.

For large-scale software development projects specifically, it is interesting to look at one of the challenges mentioned by Dikert: coordination challenges in multi-team environment. Dikert et al. (2016) describe this field as prominent and difficult, and it includes multiple challenges. As coordination problems is seen as a prominent area of challenge for large-scale agile development projects, it will be investigated further in part 2.5 of this thesis.

2.5 Coordination

Coordination is a part of any work that involves collaboration. The field has been researched from different perspectives. All elements from an organizational level, down to task level can form dependencies, which creates coordination challenges. In order to solve these dependencies, different coordination mechanisms can be used.

As presented in section 2.4.4 there are multiple challenges in regards to large-scale agile software development specifically. This subsection will cover coordination theory and provide context for further investigation of inter-team coordination in large-scale agile development projects.

2.5.1 Introduction to coordination

Thomas W. Malone identified the need for coordination theory to be used for human organizational matters, as well as for developing computer systems for coordinating activities (Malone, 1988). Malone defined coordination as "the additional information processing performed when multiple, connected actors pursue goals that a single actor pursuing the same goals would not perform" (Malone, 1988, p. 5). To be able to perform coordination, one would need two or more actors that perform tasks in order to reach a goal. In 1994 Thomas Malone and Kevin Crowston simplified the definition to be "Coordination is managing dependencies" (Malone et al., 1994, p. 90).

A dependency is when something relies on something else for support (Merriam-Webster, 2019). Accordingly, an actor is simply depended on something else to be able to take further action. To be able to identify the different kinds of dependencies, one has to consider what roles and elements are involved in a dependency, and how these elements are connected. Crowston (1994) created a taxonomy to identify the characteristics of a dependency. This taxonomy is based on the elements found in the framework created by Malone and Crowston (Malone et al., 1994). These elements are: goals, activities, actors and resources. For the taxonomy, Crowston(1994) further divided these elements into two groups: resources and tasks. Actors and resources are both considered resources, while activities and goals are considered tasks. Actors try to reach a goal by doing activities, and these activities either create- or are in need of resources to be performed. Thus, dependencies are formed as a result of the interactions and relations among these elements. The taxonomy shows how tasks and resources can be depended on each other, either with a single dependency or multiple dependencies among them. Dependencies or interdependencies can also exist within tasks or within resources, but this has not been covered by this taxonomy.

In order to manage these dependencies, one has to identify what kind of dependencies exists. There has been identified three types of dependencies (Zlotkin, cited in Malone et al. (1999)):

- Flow dependencies: activity uses resources that was produced by another activity
- Sharing dependencies: multiple activities use the same resource
- Fit dependencies: multiple activities produce a resource together

These are the types of dependencies that emerge between activities and resources, and are illustrated in Figure 9.

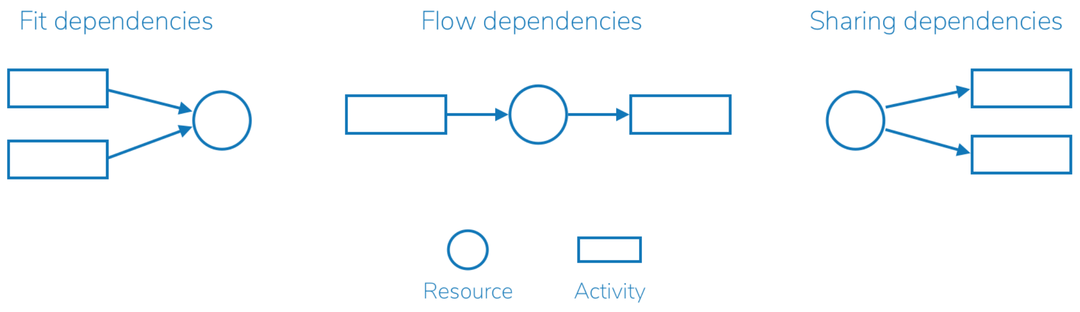


Figure 9: Types of dependencies. Adapted from Zlotkin, cited in Malone et al. (1999).

Dependencies must however not be mistaken for being the same as an interdependency. An interdependency is defined as when multiple organizations, or in this context- actors, need to consider each others needs to reach their goals (Litwak and Hylton, 1962). Thus, an interdependency can be seen as a more complex form of a dependency. While a dependency is a one-way relation, an interdependency is a mutual relation. Thompson (1967) describes three kinds of interdependencies that arises when teams are working on separate parts of a project or organization, but towards a common goal: *pooled interdependencies*, *sequential interdependencies*, and *reciprocal interdependencies*. Pooled interdependencies can be seen as an indirect interdependency. It arises when a common goal of a project or organization relies on the contribution of all involved teams, but the teams do not need to work directly with each other to reach the goal. If one team fails with their contribution, the project or the organization will also fail. Sequential interdependencies are like pooled interdependencies, but there is a direct interdependency between the involved teams. Reciprocal interdependencies is the most complex form of interdependence. The output of each team, will become input for the other teams, which makes for a tight coupling between the teams. Different kinds of interdependencies creates a need for different kinds of coordination mechanisms. For pooled interdependencies, rules and standardization is recommended, as there is little or no communication involved. Sequential interdependencies are more dynamic and needs more plans and schedules, along with some communication. Reciprocal interdependencies are in need of more communication and mutual adjustment. This implies that the more complex the interdependency is, the more coordination mechanisms needs to be present.

2.5.2 Related research on coordination within large-scale agile software development projects

Coordination in large-scale projects is necessary both among the teams, and between the team and the surrounding organization (Dingsøyr and Moe, 2014). As mentioned in section 2.4.4, coordination of teams in large-scale agile projects is considered a prominent challenge.

Dikert et al. (2016) found that using agile methods worked well within the teams, but collaboration with other teams and the surrounding organization was difficult, and dependencies were difficult to manage. This may be due to how agile methods pay more attention to activities and interactions inside a team, rather than the interactions between a team and its environment. Dikert et al. (2016) also found challenges in regards to autonomous teams. Autonomous teams gives the individual teams a lot of freedom in terms of deciding how they want to work. Dikert et al. (2016) found that some teams became too independent and focused only on their own goals and did not take the other teams and the organization into account. As each team set their own Sprint duration, it caused delays. Challenges in regards to autonomous teams has also been discussed by Stray et al. (2018b). Autonomous teams struggle with not having clear goals, lack of coaching, distrust within the team, variations in norms, and dependencies to others. When these challenges are faced at a large-scale, the dependencies also increase. Stray et al. (2018b) states that the complexity of coordination within and outside autonomous teams increase exponentially.

Another challenge discovered by Dikert et al. (2016) was to achieve technical consistency. It was reported difficulties in integrating the parts from the different teams into the system. Together with coding dependencies and different coding styles, the architecture became more fragile. The difference in the ways the teams worked also created mistrust between the teams. The coordination problems were attempted solved by trying to reduce dependencies across teams, but these dependencies were naturally a part of the project and could not be resolved. Traditional approaches caused increased workload and reduced flexibility for the teams. Scrum-of-Scrums reportedly worked for some teams, but not scaled to a global level.

A study by Paasivaara and Lassenius (2014) supports the fact that Scrum-of-Scrums may not work when scaled too large.

In addition, "Different approaches emerge in a multi-team environment" was mentioned in 21% of the investigated projects. Different agile implementations within the teams makes it harder to relocate people and cause friction within the teams. Traditional methods and agile methods were also practiced at the same time among the teams, making collaboration difficult throughout the organization (Dikert et al., 2016). Such inconsistency in agile approaches among the teams could create interdependencies and hinder coordination.

Based on the challenges already mentioned, a need for further research into inter-team coordination and how related challenges should be solved, is recognized as a necessity. In *Towards principles of large-scale agile development* (Dingsøyr and Moe, 2014), inter-team coordination is also mentioned as a topic of high priority on the research agenda.

To be able to understand and solve the coordination challenges in large-scale agile development projects, one needs to identify dependencies that arise in such an environment. Strode (2016) has conducted research specifically in regards to coordination in agile software development projects, and has created a taxonomy to identify dependencies that occur within such an environment. It must be noted that this taxonomy is not created for large-scale projects, but Strode (2016) presented this as an area of possible future studies. Stray et al. (2019) used this taxonomy for their case study on how dependencies are managed in large-scale agile projects. They were able to identify dependencies and related coordination mechanisms, and found the taxonomy useful. This suggests that the taxonomy by Strode (2016) may be a better choice when identifying dependencies in large-scale agile projects, than using the taxonomy by Malone et al. (1999). The dependencies presented in Malone et al. (1999) is not made specifically for software development projects, and may be a too strict approach for an agile environment.

Strode et al. (2012) has created a theoretical model to see how different strategies affects coordination effectiveness. The model propose three different coordination strategies: synchronization, structure, and boundary spanning. Each strategy consists of different coordination mechanisms for dependency management, and will result in either implicit coordination effectiveness or explicit coordination effectiveness. The model is illustrated in Figure 10.

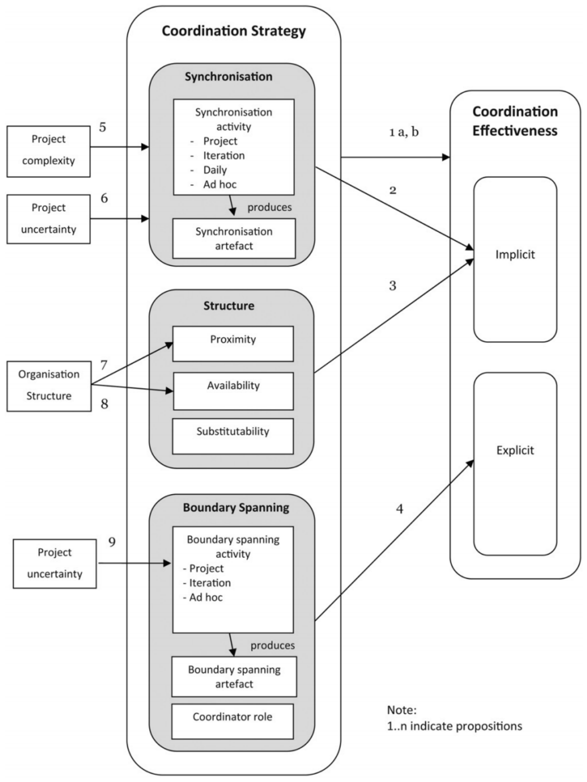


Figure 10: Coordination model for agile development projects (Strode et al., 2012, p.1230)

The model can also be used in combination with the taxonomy to link dependencies in agile development projects with coordination strategies (Strode, 2016, p. 34). However, the model by Strode et al. (2012) focus on coordination with one team involved and the interactions within the team, and does not consider inter-team coordination.

2.6 Determinants of Coordination Modes by Ven et al. (1976)

Ven et al. (1976) looks at coordination mechanisms from an organizational perspective, but on team level, which makes the theory suited when studying inter-team coordination. The article presents three modes that are used to coordinate activities: impersonal mode, personal mode, and group mode. These coordination modes draws similarities to the interdependencies identified by Thompson (1967). The study further presents factors that determines when one, or a combination of different modes are used.

2.6.1 Impersonal mode

Impersonal mode is coordination by plans, schedules, rules, formal policies and procedures, and standardized information (Ven et al., 1976). It can also include the use of technical tools. Impersonal mode requires little communication outside these activities (Boos et al., 2011). The use of coordination modes for inter-team coordination in multi-team projects has been researched by Dietrich et al. (2013). The study identified coordination mechanisms for each coordination mode. Coordination mechanisms found for impersonal mode is presented in Table 5.

Table 5: Coordination mechanisms for impersonal mode by Dietrich et al. (2013).

Coordination mechanisms for impersonal mode
• Process documentation and information in intranet • Use of documents • IT tool to follow execution process • Reporting system • Project plan as integrative map • Functionality reports and testing documents • Common database • Sales plans • Schedules

2.6.2 Personal mode

Personal coordination mode relies heavily on communication and in-person feedback. The communication happens through either a vertical- or horizontal channel. A vertical communication channel is hierarchical with communication happening between a manager or supervisor and a sub-ordinated person. Horizontal communication can be seen as non-hierarchical and happens among team members who is on the same ordinated level. Examples of personal coordination mode mechanisms found by Dietrich et al. (2013) are shown in Table 6.

Table 6: Coordination mechanisms for personal mode by Dietrich et al. (2013).

Coordination mechanisms for personal mode
• Direct contacts face-to-face or by phone
• Direct contacts via email
• Use of same resources in several teams
• Project manager's participation in teams' work
• External consultant as liaison, facilitators as liaisons
• Meeting between function head and team leaders (individually)

2.6.3 Group mode

Within group mode of coordination, mutual adjustments through meetings are used as coordination mechanisms. Meetings can be divided into two types: scheduled and unscheduled. Scheduled are meetings that are planned in advance and part of a routine, while unscheduled meetings are meetings that are not planned in advance. Unscheduled meetings are typically informal and unprepared, and are being held whenever work-problems suddenly arise. Dietrich et al. (2013) found that this is the most common mode for large-scale projects. Identified coordination mechanisms for group mode is presented in Table 7.

2.6.4 Determinant factors for coordination modes

Ven et al. (1976) found that the use of the different coordination modes depends on three factors: task uncertainty, task interdependency, and unit size. Task uncertainty can be measured as the complexity of the tasks, how time consuming the tasks are, and how easy

Table 7: Coordination mechanisms for group mode by Dietrich et al. (2013).

Coordination mechanisms for group mode
• Weekly status review meetings
• Coordination group meetings
• Inter-team meetings (informal)
• Management board meetings
• Information sharing through colocation (open space office)
• Facilitator network meeting
• Kick off meetings
• Delivery approval workshops
• Quality group meetings
• Discussion group meeting (debriefing)
• External network meetings

it is to predict the outcome of the tasks. The paper suggests that as task uncertainty increase, impersonal coordination mode becomes less suitable, as more complex tasks often requires more communication and collaboration. Task interdependency is measured by how dependent tasks are upon one another and the degree of how easy one can split the tasks into independent tasks. As mentioned in 2.5.1, one can find a hierarchical structure among the types of interdependencies. Ven et al. (1976) refers to Thompson (1967) and points out that such a hierarchy also seem to exist among coordination modes and determinant factors. Thus, increased task interdependency creates for a greater need of communication and mutual adjustment. Unit size is defined by the number of people in a work unit. In the context of large-scale agile development, unit size will refer to the number of people in each team. Unit size may affect coordination in multiple ways. Research has found that increased unit size cause decreased group cohesion and sub-groups are likely to form, each member participate less, impersonal and mechanical methods are used to spread information, complexity in demands increase, and the leadership styles become more directive (Ven et al., 1976).

Based on these factors, Ven et al. (1976) presented three hypotheses on how they may affect coordination modes:

Table 8: Hypotheses on how task uncertainty, task interdependence and unit size may affect the different coordination modes by Ven et al. (1976).

Task uncertainty	**Task interdependence**	**Unit size**
Increases in the degree of task uncertainty for an organizational unit is associated with 1. a lower use of the impersonal coordination mode 2. a greater use of the personal coordination mode 3. a significantly greater use of the group coordination mode	Increases in work flow interdependence from independent to sequential to reciprocal to team arrangements will be associated with 1. small increases in use of impersonal coordination mechanisms 2. moderate increases in use of personal coordination mechanisms 3. large increases in use of group coordination mechanisms	An increase in work unit size is associated with 1. a decrease in use of group coordination 2. an increase in use of personal coordination 3. a significant increase in use of impersonal coordination mechanisms

These hypotheses suggests that a higher level of task uncertainty, task interdependency, and unit size, either decrease or increase the needs for coordination mechanisms within the different coordination modes. Applying these hypotheses to a large-scale agile development project, can help with mapping the challenges and needs related to inter-team coordination in such projects.

3 Research Methods

This chapter will contain information on the research methodologies used for this thesis. Part 3.1 will present how the literature review was conducted and how it formed the research questions. Part 3.2 will explain the choice of research strategy, along with a presentation of the case selected for this thesis. Part 3.3 will elaborate the data generation methods and how data were collected for this thesis. The choice of data analysis method will also be explained. Lastly, an evaluation on the research methods will be presented.

The research process consists of steps and components that needs to present in order to conduct research. Figure 11 shows an illustration of the research process.

3.1 Literature review

A literature review is conducted at the beginning of a research process to gain information about a field (Oates, 2006). When getting to know a field, one can find information that is interesting to investigate further, or find areas that has not been researched yet. This creates the foundation for formulating a research question. Doing a literature review can also help supporting the researcher's claims (Oates, 2006).

3.1.1 Keywords

To find relevant literature, it is important to define key words that the literature should contain. The decided field of topic for this thesis is large-scale agile development, therefore key words related to this topic were chosen. After it was decided to further investigate inter-team coordination in large-scale agile development projects, key words related to this were added to the table. An overview of these key words is presented in Table 9.

Table 9: Key words used for literature search.

Key words
Software development, agile, agile methods, scrum, large-scale, scaled agile framework, scaling agile, coordination, inter-team coordination, coordination among teams

3.1.2 Databases

When searching for literature, online academic databases were used. Table 10 shows an overview of the databases.

Table 10: Databases used for literature search.

Databases
• IEEE Xplore
• Scopus
• Science Direct
• Web of Science
• ACM Digital Library

3.1.3 Search strategy

The search strategy used for finding literature in this thesis started out as a systematic database search to gain as many results as possible. The keywords presented in Table 9 were either used separately or combined by using AND and OR operators. To find sources that contains the exact terms, key words were used with double quotation marks, for example *"large-scale agile development"*. This would make sure the databases retrieve results that only contain that exact term. When a source was decided to be included in the thesis, a snowball strategy was used. The snowball strategy is based upon using a source to find new sources. Each paper was examined using *forward snowballing* and *backward snowballing*. Backward snowballing is to examine the reference list of a paper by looking at titles, publishers and authors, while forward snowballing is to examine the citations of a paper (Wohlin, 2014). These processes are done back and forth until all papers are either included or excluded.

3.2 Research strategy

When doing research, one has to choose a strategy on how it will be conducted. As illustrated in Figure 11, there are six common strategies: survey, design and creating, experiment, case study, action research and ethnography (Oates, 2006).

Survey: Surveys are used when gathering large amounts of data, in a structured way. The data can be used to create statistics.

Design and creation: Design and creation is used when the researcher are creating IT artifacts, such as models, methods, concepts or a full working system.

Experiment: Experiments try to either falsify or verify hypotheses, by measuring the "before" and "after", and compare the results to see how a test has affected the measures.

Case study: Case studies investigate an instance of a topic, and aims to give an in-depth understanding of the selected topic.

Action research: Action research are not necessarily focused on textual theories and models, but rather problems connected to real-life situations. When conducting action research one plan the actions and perform them in real-life situations, and reflect on the outcome.

Ethnography: Ethnography research are used when studying cultures and groups of people. The research is often conducted by doing interviews and observations of people in their environment, to get an detailed insight

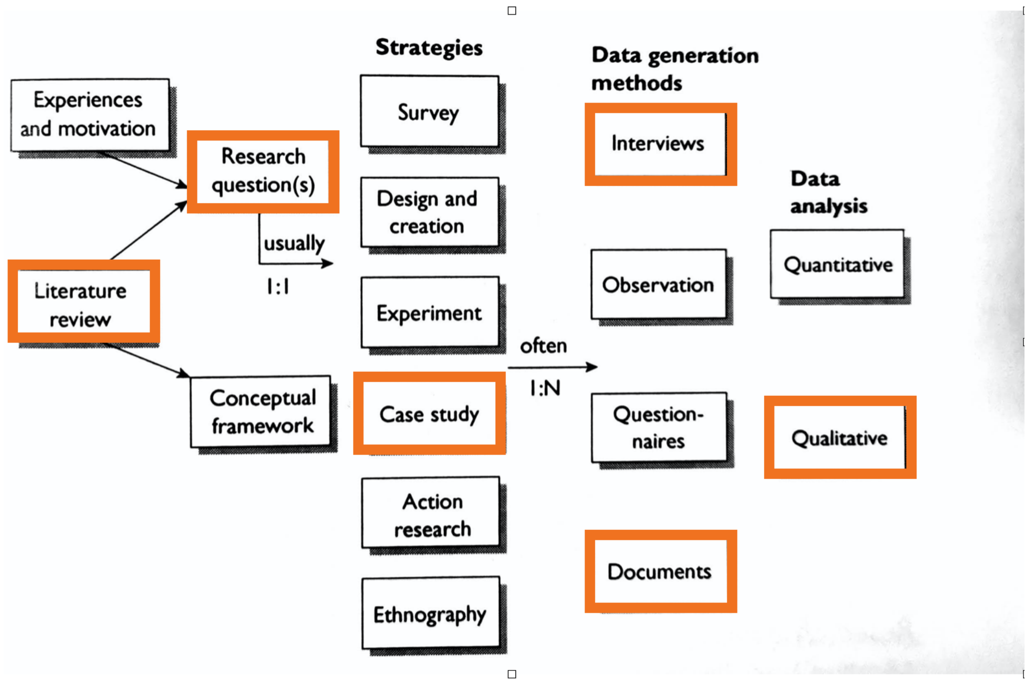


Figure 11: Illustration of the research process (Oates, 2006, p. 33), with highlights on the chosen strategy for this thesis.

3.2.1 Case study

A case study is used when conducting in-depth research of an instance of a topic (Oates, 2006). Case studies study connections and relations within the chosen case, and how these relations affect each other. A case study is also performed in the natural environment of the case, and not in an artificial setting. Both qualitative and quantitative data can be used, thus multiple data collection methods are applicable for this research strategy. There are three different types of case studies: exploratory, descriptive, and explanatory.

An exploratory case study are used when the a topic lacks preliminary research and one aims to acquire more insight into the topic. This type of case study try to uncover what is going on and how things happen. The results from an exploratory case study creates a basis for future research and helps define new hypotheses or research questions. A descriptive case study is used to generate descriptive information about a specific topic. It does not address casual relationships. This type of case study simply provides more knowledge or information about a topic, and can not necessarily be used to develop a research question. An explanatory case study aims to explain why things happen, by finding connections among multiple factors and comparing the results to different findings.

This thesis aims to get an in-depth insight into the topic of large-scale agile development projects with focus on inter-team coordination, and contribute as basis for future research. Studying a real-life large-scale agile development project can provide the information needed to gain insight on this topic. Runeson and Höst (2009) also states that using a case study as research strategy is well suited for software engineering research, as one can achieve a deep understanding of the studied topic. Therefore, an exploratory case study has been chosen as research strategy.

3.3 Data collection

Data collection is used to gather information for research purpose. The collected data can either be *qualitative* or *quantitative* (Oates, 2006). Qualitative data is data that consists of descriptions, sounds, images, videos. Quantitative data is data based on numbers and can easily be measured. There are four common data collection methods: interviews, observations, questionnaires, and documents. When gathering quantitative data, questionnaires are the most suited as it allows to ask close-ended questions or use multiple choice, which can easily be generated into numbers and analyzed. Interviews, observations, and documents are suited methods when gathering qualitative data, as these methods do not produce numeric data and are good for exploration and getting in-depth information on a topic. The different collection methods can also be combined to obtain different perspectives and more details.

3.3.1 Interviews

An interview is a planned conversation with the purpose of gathering information on a specific topic (Oates, 2006). In a research context, the researcher is responsible for conducting and leading the interview. The interviewee should be informed about the topic of conversation, the purpose of the research, and agree on that the information gained will be used for research.

If the interview is planned to be recorded, the interviewee must also be informed about this, and agree to being recorded. Interviews are good for obtaining in-depth information, and is therefore a much used data generation method for case studies (Oates, 2006). There are three types of interviews: structured, semi-structured, and unstructured (Oates, 2006). Structured interviews are planned in advance, and uses ordered identical questions for each interviewee. Such interviews are effective and a good way to get clear answers to your questions, but lacks flexibility and one might miss out on in-depth information. Semi-structured interviews are planned in advance, with prepared topics and questions, but allows for changing the structure, reformulate existing questions or add new questions during the interview. This type of interview is more flexible, and lets the interviewee speak more freely about the topic. This is good for collecting more in-depth information, and at the same time making sure that your planned topics and questions are covered. When conducting unstructured interviews, a topic is introduced and the interviewee gets to talk freely about the topic. It is not planned in advance and questions arise depending on the where the conversation heads off. In such interviews one can get more insight into the interviewee's feelings and thoughts, which can broaden the perspective of the topic. On the other hand, it can be time consuming and the interviewee might share information that is not relevant.

For this thesis a semi-structured interview was chosen. This is due to the need for exploration on the topic and gathering in-depth information. With limited time for each interview, as well as specific topics the interviewer wanted to cover, it was necessary to have some structure.

The interviews used as data for this thesis was conducted by the thesis author and by researchers from SINTEF and the University of Oslo. Master's theses have a limited time scope to conduct data collection, but the case study in this thesis is based on information collected through a period of three years. Being able to combine data collected by the thesis author with data collected by researchers at SINTEF and the University of Oslo strengthens this case study. It could have strengthen this case study more if one were able to do observations in addition to interviews. However this was not manageable for this case study, as the investigated project was finished before this case study was conducted.

The first interview round consisted of 13 interviews that were held in December 2017. The second interview round consisted of 10 interviews conducted in January 2019. The last round of interviews were held in the period of November 2019 to February 2020, and consisted of 14 interviews. The thesis author was present during the interviews in November 2019 and January 2020, and also took part of the transcribing afterwards. The interviews held in February 2020 were conducted by the thesis author with the purpose of gathering information specifically related to inter-team coordination and more details on the last project phase. Table 11 shows an overview of the number of interviews in each round, mapped to the interviewee's project role.

In total, 37 interviews were conducted, which were transcribed into a total of 507 pages. To gain a correct and detailed image of the whole project, people with different roles and distributed among as many teams as possible were chosen as interviewees. All interviews has been transcribed and anonymized, as well as password protected and stored on an encrypted disk. The interview guides can be found in Appendix A and Appendix B.

Table 11: Number of interviews per role

Round 1		Round 2		Round 3	
Number of interviewees	Roles	Number of interviewees	Roles	Number of interviewees	Roles
1	Technical architect	2	Test	3	Technical architect
1	Functional architect	1	Business manager	2	Developer
1	Developer	1	Technical support	2	Test
1	Front end lead	3	Tech lead	3	Product owner
2	Scrum master	2	Product owner	2	Business
1	Test	1	Functional architect	2	Project manager
1	Functional manager				
1	Main architect				
1	Solutions manager				
1	Construction manager				
2	Applications architect				
Total		Total		Total	
13		10		14	

3.3.2 Documentation

In addition to interviews, documentation has been used as a source of data for this thesis. The documentation consisted of project descriptions, plans and reports received from informants that has been involved in the investigated project. The documentation has been used to gain insight about the project and to describe the case as it is presented in section 4.

3.4 Data analysis

Based on the collected data, a case study database was created. This is also encouraged by Yin (2009), as a way of increasing reliability and maintaining a chain of evidence. This will be further elaborated in section 3.5.2. The database was created by using QSR International's NVivo 12, which is a qualitative analysis software tool. Transcribed interviews were imported into the software and then thoroughly examined to find relevant information. The interviews was then coded into nodes represented by different themes. Quotes and information on a specific topic were found by looking through information that has been connected to the nodes. By doing this one can easily process the data into useful information. Furthermore, the information extracted from NVivo was applied to the theoretical model of coordination modes by Ven et al. (1976).

3.5 Evaluation

This part will contain an evaluation of the validity and reliability in regards to how the research for this thesis has been conducted.

3.5.1 Literature review

The literature review conducted in this thesis is based upon multiple- and different types of sources, such as journal articles, books, websites and conference papers. The sources has been found by using academic databases such as IEEE Xplore, Scopus, Science Direct, Web of Science, and ACM Digital Library. These are databases that contain academic publications, and are reliable choices for finding literature.

When choosing sources, the thesis author considered what type of source to include in the thesis. Journal articles are considered to be a reliable type of source, as most journal articles has been through a peer review. This means that it has been read and evaluated by other academics, and it has to be of high quality in order to be published into a journal. Most of the sources used in this thesis are journal articles.

Books can be considered to be reliable sources, but it must be noted that not all books may be peer reviewed. Some of the books used in this thesis are published by Springer and SAGE, which use peer reviews for their publications and are considered to be scientific publishers.

Conference papers are often not as detailed as journal articles, as they are to be presented in a conference. Most conference papers within the field of software engineering are peer reviewed before being published, which strengthens the reliability for conference papers within this field.

Websites may not be a reliable source, as they can be easily changed. It is therefore important to note in the references the date the website was accessed. If the information found on a website gets edited, the reader will know that the edit has happened after the website was cited. Websites are not much used in this thesis, but has been used when finding definitions of single words, and to get information about Scrum and agile development. The Agile Manifesto was first published by the original authors on its own website, and the Scrum Guide is also originally provided through a website. As these can be considered to be the original source of information, it was decided to be reliable enough to be used for this thesis. One of the sources used in this thesis is from the Deloitte's company website. Deloitte had conducted a survey that shows the use of agile methods from 2002 - 2017. The website does not provide detailed information on how the study was conducted and may not be a reliable source. This was also mentioned in section 2.3. This is also supported by Stavru (2014), who has criticized industrial surveys on agile methodologies. Stavru (2014) found that such surveys often lack details on research methods, and evaluation of the study's validity and reliability.

This thesis have also used literature from practitioners, such as Kent Beck, Ken Schwaber, and Jim Highsmith. Practitioners are not researchers, and focus on providing information to other practitioners within the field. Thus, they are not bound to deliver literature by the same standards as researchers. Therefore one should be critical towards using such literature to support your own work. However practitioners have real-life experiences and might provide valuable insight. The practitioners cited in this thesis are well-known practitioners and frequently cited within the field of software engineering, and were therefore included in this thesis.

When choosing sources, the thesis author also looked at the number of citations and tried to find the articles with the most citations. A lot of citations means that a lot of people has found the article useful and reliable. However the author has also considered less cited articles. Some articles are quite newly published and therefore may not have as much citations yet. In addition, the topic of large-scale agile development is not much investigated yet, which can cause the number of citations to be low.

3.5.2 Case study

Yin (2009) describes four tests that can be used to evaluate the quality of a case study: construct validity, internal validity, external validity and reliability.

Construct validity

The purpose of this test is to test whether the chosen measurements covers the investigated topic. Thus, one need to choose the right measurements to gain relevant data and to correct represent the investigated case. To ensure construct validity, two steps should be covered (Yin, 2009, p. 42):

1. *define neighborhood change in terms of specific concepts (and relate them to the original objectives of the study) and*

2. *identify operational measures that match the concepts (preferably citing published studies that make the same matches).*

In addition one can increase the construct validity by using multiple sources of evidence, creating a chain of evidence, and to have key informants review the draft case study report (Yin, 2009). For this thesis, research question have been defined, and a literature review has been conducted to support findings and claims based on the case study. When collecting data, interviews and documents has been used as sources of evidence. A chain of evidence has been maintained as the findings presented in section 5 can be found in the case study database, and the information found in the database is based upon interview guides used during data collection.

Internal validity

Internal validity is a concern in explanatory case studies, when one is analyzing relationships among events and try to explain how and why something has occurred (Yin, 2009). As an exploratory case study has been conducted for this thesis, internal validity has not been concerned.

External validity

External validity covers the issue whether the research findings can be generalized and applied to other similar cases, or if it is only limited to your case (Yin, 2009). Using theory is stated as one of the tactics to increase external validity (Yin, 2009). For this thesis, a theoretical model by Ven et al. (1976) has been used.

Reliability

The purpose of reliability is to reduce inaccuracy and bias in a study. To achieve reliability, one has to document how the research has been conducted, so other researchers are able to repeat the case. This can be achieved by documenting the case study process and by creating a case study database (Yin, 2009). For this thesis, the case study process has been explained in section 3, and collected data was stored in a case study database.

3.5.3 Research ethics

All interview participants have received an information letter. This letter provides information about the research project and their participation. The participants are informed that they are allowed to withdraw from the study.

All participants have been anonymized and no sensitive data has been collected. Only information about the participant's roles and team names has been collected. This was done to get an overview of the project structure and to connect different perspectives to different roles. This may increase the understanding of the information provided by the interviewee.

4 Case

This section describes the investigated project that is used as case for this thesis. It provides background information, a presentation of the project organization, and information about how the project progressed, in chronological order, throughout the different phases.

4.1 Background

The investigated project was a part of a modernization process of the IT systems within a large Scandinavian welfare organization that provides public services. The IT systems dated as far back as the 1970s, making the administrative procedures and services provided by the organization cumbersome and time consuming, both for employees and end users. As a part of the modernization process, it was decided to create a software solution to automate the administrative procedures and make the system mostly self-serviced for the users. This process would be called the Modernization Programme. In 2011 it was decided to divide the Modernization Programme into three projects: Project A, Project B, and Project C. Project A started in 2012, but was deemed a failure after less than a year as a result of ambition levels set too high, the complexity became larger than predicted, and the project management and project organization were insufficient. About 31 million euro were calculated to be lost, and the failure gained great media attention. The Modernization Programme was shut down, and the three projects had to be replanned. Project A started over and was finished in 2016. Project B, which is investigated in this case study, therefore faced a lot of pressure on succeeding. The organization has about 19 000 employees and the services related to Project B have about 140 000 end users. Project B was estimated to cost about 90 million euro, which puts an emphasis on the size of this project.

Project B was divided into three phases: Phase 1, Phase 2, and Phase 3. In addition there was a preparation phase prior to phase 1, and an ongoing maintenance phase after phase 3 was finished. The preparation phase lasted from October 2016 to February 2017. Phase 1 started in February 2017 and lasted until December 2017. Phase 2 started in September 2017 and finished in November 2018. Phase 3 started in September 2018 and was finished in June

2019. Figure 12 shows an illustration of the project timeline. Throughout the project the methodology has gone from working by a "water-scrum-fall" methodology, then a bi-modal phase, while the last part of the project was characterized by agile methods, cross-functional autonomous teams and continuous deliveries. Such big changes in working method within a large IT project is not common, which makes it a highly interesting case to study in regards to large-scale agile development.

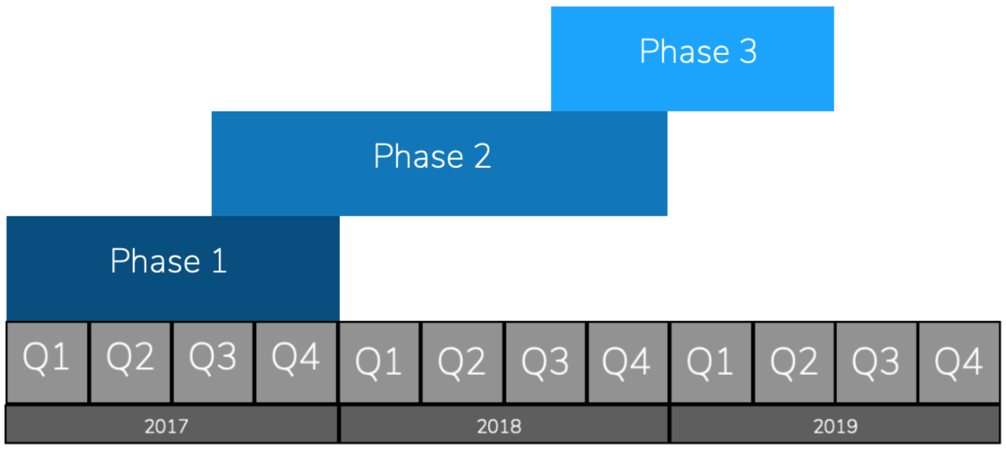


Figure 12: Illustration of the project timeline

4.2 Project organization

Along with staff from the welfare organization, two external contractors were hired. One contractor was hired to work with development, while the other contractor was hired to work on the business part of the project. The project started off with one development team and expanded to two teams during Phase 1. By the end of this phase and during the start of Phase 2, the number of teams increased to five. Each team had 9-10 members and consisted of a Scrum master, 1-2 architects, 1-2 testers, and 5-6 developers. An illustration of the organizational project structure at the time is illustrated in Figure 13. In Phase 3, the numbers of teams increased to 10. At this point the teams became cross-functional and the business roles should also be present within the teams, which increased the team size as well. The teams now consisted of a business advisor, product owner, domain expert, functional architect, tech lead, 4-5 developers, tester, and a UX designer. The organizational project structure for Phase 3 is illustrated in Figure 14.

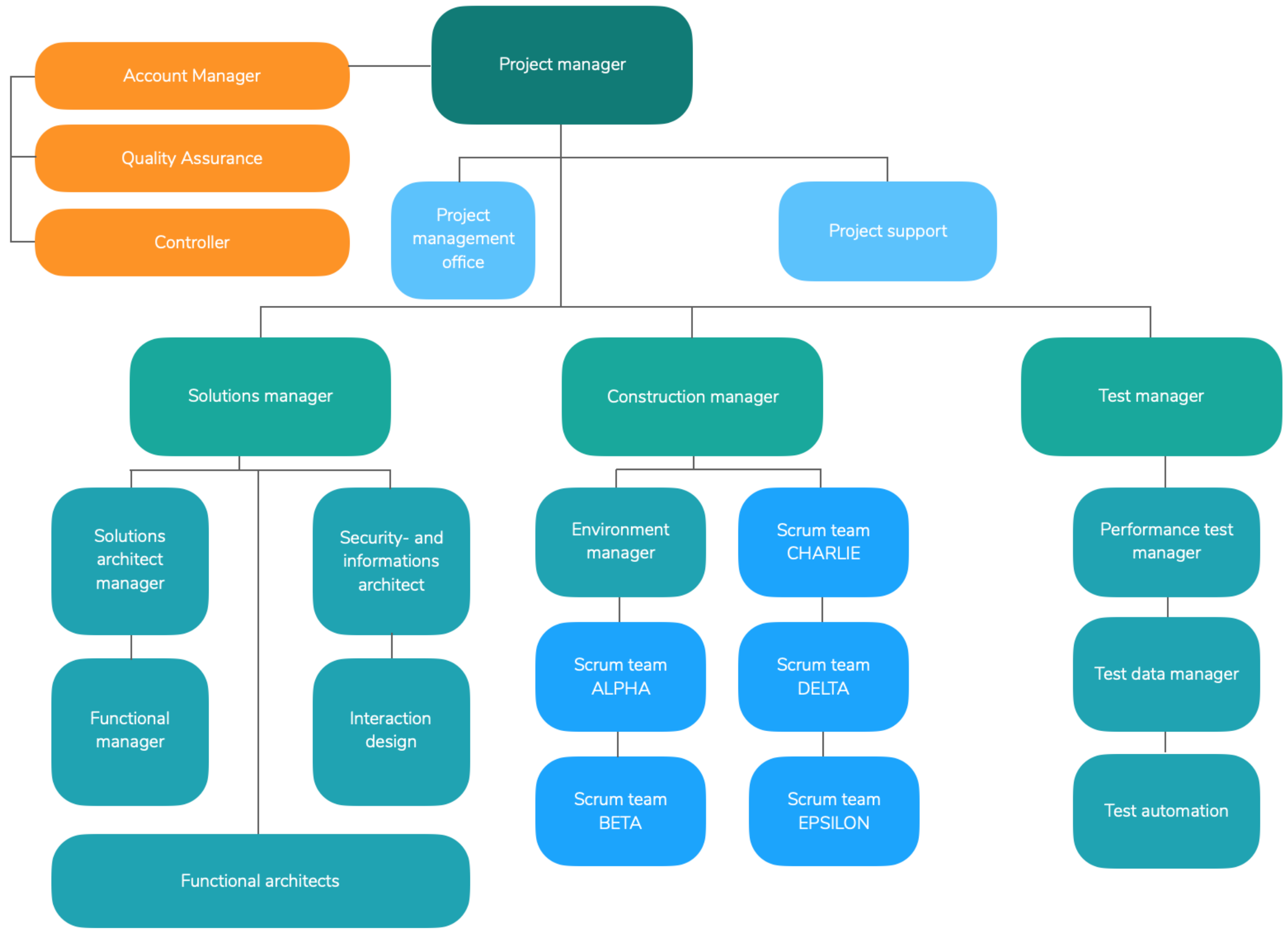


Figure 13: Illustration of the project organization in Phase 1, November 2017

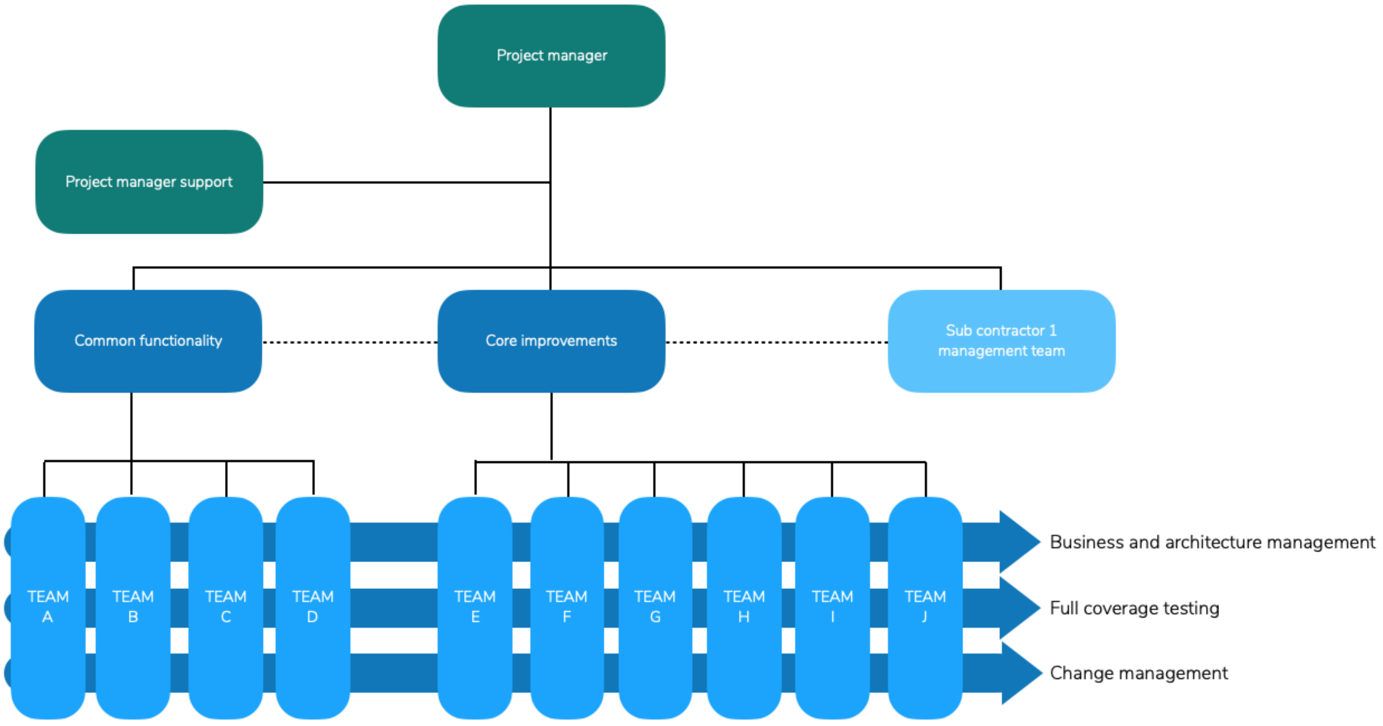


Figure 14: Illustration of the project organization in Phase 3, September 2018

4.3 "Water-scrum-fall" methodology

The "water-scrum-fall" methodology used by the investigated project was created by the welfare organization itself and is based upon PRINCE2. The methodology consists of 6 main phases, illustrated in Figure 15. While the first three phases lean towards a more plan-driven way of working, the constructing phase is influenced by agile methods. Work is being done in iterations, and the teams use activities and events from the Scrum methodology.

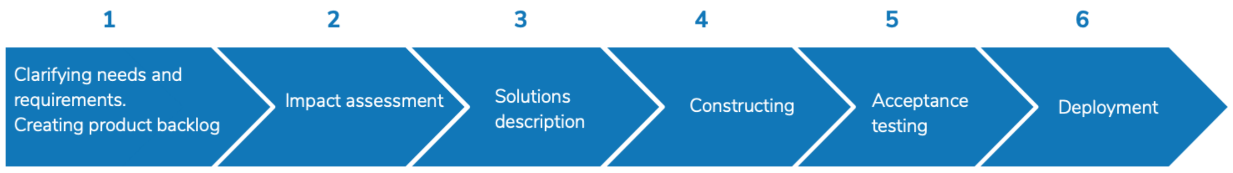


Figure 15: Phases of the "water-scrum-fall" methodology

4.4 Project summary

The first phase aimed to produce the first automated version of a small administration procedure. This was a less complex delivery, and the solution was successfully delivered as planned in December 2017. This phase worked by the "water-scrum-fall" methodology presented in section 4.3.

Phase 2 started in September 2017 with five teams, with the purpose of creating a solution for automating a larger administration procedure. This was a far more complex solution as it would affect executive officers and supervisors from the welfare organization, employers and end users. In addition, the system had to fulfill laws and regulations set by the government. A lot of the software developed in this phase was also planned to be re-used for similar future projects within the welfare organization. The working methodology at the beginning of Phase 2 was still the "water-scrum fall" methodology, but the plan-driven way of working was more distinct and present than the agile methods. A lot of time was spent on documentation, planning, handovers and impact assessment, which also created a higher level of dependencies in the project. At the same time, progression and value creation was low. It was decided to put together a cross-functional team, Team X, that would work with a low-risk part of the project, and they were allowed to work as agile as they wanted to. This made the project *bi-modal* - a parallel of teams where one part of the teams worked by the old methodology, while the new team worked by a high level of agility and autonomy.

Team X started to deliver value, and other project participants also wished for a higher level of agility in the rest of the project. After a year, the project manager put together a new team, Team Y, consisting of people from both the IT part and the business part of the project. This team was given the task to advise the project manager on how to proceed with the project. Team Y suggested to completely change the working methodology and start working with agile, cross-functional- and autonomous teams, with continuous deliveries. It was a big risk to take, as there were 200 people involved in the project and only four months left until one of the most important deliveries.

The project manager decided to follow up on the advice, and the working methodology and project structure were changed. Middle management and administration were cut down, while new cross-functional autonomous teams were formed in Phase 3. The project went from having quarterly deliveries to daily valuable deliveries. The project was successfully finished in June 2019, and also received a digitalization award.

5 Results

This section presents findings from the case study by using the coordination model by Ven et al. (1976). Thus, the findings will be divided into three categories: impersonal mode, personal mode, and group mode. Determinant factors for coordination modes will also be considered. The aim is to uncover the use of different coordination mechanisms used throughout the investigated project, and see how the change in agile methodology affected inter-team coordination. Due to overlapping phases in the investigated project, it has been decided to merge the three original phases into two phases. The first project phase and the second project phase, as they were described in section 4, will be referred to as **Phase 1** from now on. The third project phase, as described in section 4, will now be referred to as **Phase 2**. By comparing these two phases, one can see how the use of coordination mechanisms changed and how it affected inter-team coordination.

5.1 Impersonal mode

Coordination mechanisms found for impersonal mode is presented in table 12.

Table 12: Coordination mechanisms found for impersonal mode

Phase 1	**Phase 2**
• Documentation and planning • Jira • Working area • Whiteboard	• Documentation and planning • Jira • GitHub • Working area • Whiteboard

Documentation and planning

Confluence was used as wiki for the project, and all information and documentation was added there. Confluence is defined as: ".. a collaboration wiki tool used to help teams to collaborate and share knowledge efficiently" (© Atlassian, Inc., 2020a). This tool helped the teams get an overview of the project and to find necessary information.

Confluence was used for information and documentation sharing for both Phase 1 and Phase 2. However, the documentation quantity was somewhat reduced for Phase 2, as they were working more agile, with autonomous teams, rather than spending time on documentation. When asked about the biggest challenge from Phase 1, one of the technical architects responded:

"We spend too much time and energy on solution descriptions. Team architects spend too much time on solution descriptions, rather than communicating with the team." ... "They (Customer) have a lot of requirements for the solution descriptions. They use their own methodology ("water-scrum-fall"), and that requires a lot of documentation for each solution description. Not necessarily documentation that are being used by the teams as one would like to. You are not free to do what is best for the team. You have to follow their methodology, and not being able to change things on the go is a challenge." - Technical architect

In Phase 2, it was stated to be a lot less documentation and planning, and more freedom to prioritize what to do at the moment:

"..Earlier it was a lot of documentation, such as solution descriptions and a lot of planning ahead. After changing working methodology it is a lot less documentation and we work on smaller fragments of the solution." - Scrum master

"We documented a lot in the first phase, both solution descriptions, estimates and everything. That is something we have moved away from... We were more dependent on planning in the beginning of the project, but later it became more important to look at what is necessary right now, which lowers the risk, so one could do what is most important at the moment." - Technical architect

Jira

Jira is a project management tool used for software development projects (© Atlassian, Inc., 2020b). In the investigated project, Jira was used to manage tasks, user stories and issues. Jira was used in both Phase 1 and Phase 2.

During Phase 1, the work with tasks, user stories and impact assessment were clearly divided into Phase 1 and Phase 2 of the "water-scrum-fall"-process, and then added to Jira. Requirements and user stories were detailed, but the lack of communication among the people working in the different phases of the "water-scrum-fall" methodology caused handovers and misunderstandings with the developers.

In Phase 2, this somewhat changed. As the working methodology had changed, it was no longer important to follow the "water-scrum-fall"-process as rigidly as earlier. In addition, user stories and requirements became less detailed. Jira was still being used, but as a result of the change in working methodology, the information added to Jira was less detailed. This gave developers more freedom to work as they preferred and to add functionality they found necessary.

In the beginning dependencies was documented only in Confluence, and not in Jira.

"One of the challenges regarding dependencies is that it has only been documented in Confluence and not in Jira, which is our common working tool. This has caused the dependencies among user stories to become less clear." - Construction manager

As this was found to be a problem, they started to add dependencies to Jira as well. When asked if all dependencies are now found in Jira, the construction manager said:

"I don't know if this has been fully completed yet, but the instructions should be clear. You have functional dependencies and technical dependencies, and the technical dependencies are the most difficult to reduce, especially in other teams." - Construction manager

In addition to using a regular whiteboard, Jira was also used as a board to manage tasks and give team members an overview over current prioritized tasks.

"We agree within the team that Jira is master, that it must be updated, and then I try to make sure that the whiteboard is updated as well. One needs to take into account that if someone has to work from home or other places, they would also have access to the updated status." - Scrum master

Thus, Jira was mostly used for the same reasons during both phases, but the information added to Jira in Phase 2 was less detailed. In addition, dependencies was also added to Jira in Phase 2.

GitHub

One of the new tools that were introduced in Phase 2 was GitHub. GitHub is a software development platform that is used for hosting and reviewing code, task management and version control. It allows you to easily collaborate with others on software development.

"So much has changed the last year. All code was stored internally earlier, but is now moved to GitHub, so everything is public now. It was a big transition in how you worked, with technology, coding and stuff. Now you deploy everyday, rather than once in a while. You have to be a bit more careful about mistakes, but since you deploy everyday you get more freedom as well. A small mistake can be fixed quick." - Front end developer

Whiteboard

Physical whiteboards were used for both phases. Each team had their own whiteboard at their location. This was used within the teams to keep track of important focus points and tasks during each iteration, and was used during meeting sessions such as daily stand ups and retrospective meetings.

".. after retrospectives, we have added focus points on our whiteboard, just a few

points to focus on throughout the iteration. We would remind ourselves during stand ups to focus on these points" - Scrum master

It was found that architects also use whiteboards during meeting sessions with other architects. When asked about how the architects use whiteboards, the response was:

"We use them all the time. During work meetings we sketch stuff on the whiteboard all the time. We often add input to the board, and then discuss it. It could be on the board, or we show something we've written down earlier". - Technical architect

There were not found any changes in the use of whiteboards from Phase 1 to Phase 2.

Working area

Working area has been an important factor for successful coordination throughout both phases in the investigated project. In the early beginning of the project, some teams were located on the fifth floor, while other teams were located on the fourth floor. The teams located on the same floor worked on the same part of the solution. However they still had the need to communicate with the other teams to make sure the different parts became integrated with each other. This was found to be troublesome, and one of the developers emphasized the importance of co-location as coordination mechanism:

"There has never been any big issues for me with coordination. However it was quite problematic in the beginning. We were located in fifth floor, and then the self-service teams were located in fourth floor. Then there were some technical decisions being made that could be difficult to understand for outsiders. This made communication between us and the self-service teams less efficient. However, it was not a problem for me personally. But the backend developers probably found it annoying. The communication wasn't 100%. Maybe the most important part of coordination is co-location and short physical distances. You know who

you talk to. That seems to be the key factor to successful coordination." - Front end developer

Teams that have the most dependencies among each other are placed close to each other. There has however been issues with too little space, causing the need to move some people to other locations:

"We are currently using the whole part over there, so that is the best we could get, definitely. But we have moved a lot, and that has not been... But right now we have no space left. One more, and there are no more space left. We don't have one single available chair right now. It is completely full. Two new people arrived yesterday, and we had to give one chair away. So we moved one of the managers. That is how it is." - Technical architect

One of the problems throughout Phase 1 was handovers that caused delays and were pushed into the next iteration. To better the collaboration among teams and to solve the handovers more quickly, it was decided to move functional architects closer to the construction teams:

"There is this classic problem with handovers. It has been too many of them, and at the same time, the project is quite dynamic. Our contract is an agile one, so we have plenty of room to collaborate closer with each other. We are not set to live with these handovers. So the handovers is pushed into the next iteration. So we co-located some of the functional people together with us.." - Main technical architect

"This caused less formal meetings and more ongoing cooperation" - Main technical architect

"We sit together with the construction teams, and the functional architects is now working closer to them as well. They didn't do that earlier." - Main technical architect

5.2 Personal mode

Coordination mechanisms found for personal mode is presented in table 13.

Table 13: Coordination mechanisms found for impersonal mode

Phase 1	**Phase 2**
• Instant messaging • HipChat • Skype • One-on-one conversations	• Instant messaging • Slack • One-on-one conversations

Instant messaging

Instant messaging played a big role throughout the whole project. It was being used for both as a quick way to get in touch with people, either to discuss problems, to inform about new information regarding the project, or to socialize. It was being used for both vertical and horizontal communication. Managers could use instant messaging to reach the teams and provide them with information, and anyone could directly contact the managers. This caused the hierarchy to become less prominent. In addition anyone across the different teams could also contact each other.

"We used HipChat in the beginning. Or Skype. And you could just contact people whenever it was necessary. We had channels where you could share information. It wasn't a total divide between the teams and the managers. You could basically talk to whomever you'd like." - Front end developer 1

When asked about when they changed communication tool, the response was:

"It was at the time we started with continuous deliveries. We had an increased need to talk to people outside our own teams as well at the time. And then Slack was the preferred tool." - Technical architect

This shows it was an increasing need to communicate among teams when changing working methodology. One of the other reasons for the change of tool was due to more people were familiar with Slack and preferred it over HipChat. When asked about what Slack was used for, the responses were:

"In the beginning Slack was a casual announcement place. It was used for simple and informal information such as "if you are working overtime, you can order pizza here..", and then there were news about the project, and if there were new test data everyone needed to be informed about. But over time it has become more discussions, like "This doesn't work very well, what should we do here?", and then some would come and help out with a solution. It went from being not that much used, to much more used." - Front end developer

From these findings, one can see that instant messaging has been a part of the project from the beginning, but that it has gradually become a more important tool for collaboration and solving issues across teams, especially after the change of working methodology.

One-on-one conversations

One-on-one conversations were frequently used during the whole project. Such conversations happened between team members across different teams, but also between managers and team members. These conversations were either used to solve problems, or to transfer general information. During Phase 1, each team had their own Scrum master. The Scrum master was usually the one to do one-on-one conversations with managers and other teams, and transfer information between their team and the others.

"Throughout the first phase and parts of the last phase, it was only the Scrum masters that communicated with the others. While the rest of the team had focus within their own team. But I guess it depends on each person a bit too. I usually just go and ask anyone if I need something. But on a general basis, there were some hierarchy." - Frontend developer

"I think that in the beginning, the communication happened more on a management level, and that the Scrum master talked with the other teams, while the teams just worked with their assigned tasks. But right now I feel like everyone can talk to anyone, when it is necessary." - Technical architect

In Phase 2, the amount of managers were reduced and the teams no longer operated with a designated Scrum master. This caused the communication to be less vertical and more horizontal. In addition, the communication between the teams increased, because the barrier to talk to each other was lowered.

5.3 Group mode

Table 14: Coordination mechanisms found for group mode

Phase 1	**Phase 2**
• Planning meetings • Scrum-of-scrums • Stand up meetings • Retrospective • Demo meetings • Technical architecture forum • Functional architecture forum • Tester forum • Status meetings • Technical review • Ad hoc meetings	• Stand up meetings • Status meetings • Focus meetings • Functional architecture forum • Ad hoc meetings

From Table 14 one can see that group mode of coordination has been an important coordination mode in this project. A lot of group coordination mechanisms have been reduced from Phase 1 to Phase 2, due to the change of working methodology. Most of the meetings that were being used in Phase 1, were scheduled meetings that followed the working methodology. Most of the meetings that did not follow the working methodology from Phase 1, were continued to be used in Phase 2.

Planning meetings were held at the beginning of each sprint, and all teams attended these meetings. These meetings were used to present the tasks and requirements for the upcoming sprint, as well as dependencies among the teams. These were replaced with quarterly focus meetings to discuss on a superficial level what should be focused on for the next quarter, rather than every third week.

Scrum-of-scrums, retrospective, demo and technical reviews were a part of the "water-scrum-fall" methodology used in Phase 1, and were therefore not a part of Phase 2. However stand up meetings, which was also a part of the methodology used in Phase 1, was also mentioned to be used by some of the teams in Phase 2.

In Phase 1 it was also common to have meetings (forums) among the different roles. Testers met on a regular basis to discuss testing problems, and technical architects met to discuss technical dependencies and problems with the solution, and functional architects met to discuss requirements and progress. The developers did not have any dedicated forums, other than being a part of the technical reviews. Meetings to handle dependencies happened mostly among the functional architects and product owners. From Phase 1, the functional roles had "functional forums". This was originally removed when the project entered Phase 2, but it was re-introduced as it was found necessary for these roles to have an arena to discuss dependencies among the teams.

"After Christmas we stopped with the functional forum, and we managed without it for 5 weeks, but then it was re-introduced in February." - Product owner

Stand up meetings

Stand up meetings were used on a daily basis during Phase 1. The meetings were held within each team and lead by the Scrum master. It was used as a short status meeting to present what each person within a team was currently working on, and what they would do next, and short discussions on the current status. In Phase 2, each team decided how they wanted to work and there was no requirement to do daily stand ups. As the teams had no designated Scrum master anymore, there was no one to lead the meetings either, unless someone took the initiative to do it. Due to this, stand ups were being held less frequently. It was found that it was typically held twice a week, but that it might not be as efficient as it used to be in Phase 1, and that stand up meetings might not even be necessary when working with continuous deliveries:

"We've got stand up meetings two times a week, but there have been wishes to instead focus on the daily production. Stand-up is a bit of a problem-area now. It is closely tied to how Scrum works, but when you start working differently... the stand up meetings lately haven't been very good. They used to be, but now it is a bit weird. Now we have short stand ups, and then we try to talk a bit more once a week. It is a social thing, but at the same time you shouldn't get too sidetracked, so it is a bit like... who is going to lead the stand up, when you don't have a scrum master? We tried to let the testers lead for a while, but the meetings got too focused on test. I am not sure that it is right to have traditional stand up meetings, when you work this way."
- Scrum master / Teach lead

Status meetings

It was found that it was necessary to have frequent status meetings as a common arena for all the teams. Weekly status meetings among functional architects were used in Phase 1 to discuss the requirements, along with status and progress.

After changing to cross-functional autonomous teams in Phase 2, it was no common arena for all the teams to meet and discuss the current status among the teams.

"It is something we are struggling with now. We are so many teams, and we are large teams, and now the layer that facilitates scheduled communication among the teams are removed. It was removed as soon as we changed to Phase 2. But we can see that error flow and defects that are discovered among the teams. To report a total status on what we miss is very difficult. I think they removed too much of that layer." - Scrum master

Due to the struggles with being updated on other teams status and progress, weekly status meetings were reintroduced.

"Lately we have started with Monday meetings. People on the outside struggles to understand the current status. We have a large solution, with a lot of bugs. And to coordinate all the teams outside the project, they are not interested in the eight teams working autonomous and doing things differently from each other. They want things to be predictable, with reports and current statuses. We struggle with communication outwards." - Scrum master

Functional architects from the teams and product owners were present at these meetings, and discussed current status on functionality and what is currently being delivered to keep everyone updated and increase the communication flow.

Ad hoc meetings

Ad hoc meetings were used in both Phase 1 and Phase 2, but the use increased significally in Phase 2. Due to the decrease in scheduled meetings, people rather used ad hoc meetings whenever it was needed. Ad hoc meetings were often used among people with the same roles across the teams, so they would discuss problems that were relevant for their role. In addition ad hoc meetings were held within the teams, typically among the developers to keep each other updated on what they are working on or to solve issues that arose. If two teams had dependencies among each other, a representative, typically product owners or functional architects, from one of the teams would go and speak to the other team. Thus, the functional roles experienced a larger increase in ad hoc meetings. Whenever technical problems arose, developers could also go and speak to other teams. The barrier for doing this was low and communication was more informal and much less influenced by hierarchy during Phase 2. This was also one of the factors for an increase in ad hoc meetings.

5.4 Determinant factors for coordination modes

Task uncertainty

Task uncertainty was considered to be low during both phases of the project, but somewhat increased during Phase 2. In Phase 1, the requirements were very detailed, which caused the teams to have a clear view on what should be done. In addition there was a high use of group mode, and frequent use of meetings caused everyone to be updated on current status and changes.

When asking one of the developers if they ever experienced task uncertainty, they responded:

"No, not really. We got handed the tasks at the beginning of a sprint, but it was the team's decision to decide how to do it and prioritize it. It was quite ok. It was worse with tasks that didn't seem too difficult to begin with, but was discovered to be quite hard to solve. They could easily last one or two sprints, and if it lasted too long you had to take some time to think about what to do with the task." - Frontend developer

When asked whether the teams experienced any task uncertainty, one of the technical architects responded that lack of detailed tasks caused an increase in task uncertainty among the teams.

"We think we described things a bit too detailed. The teams got used to getting very detailed descriptions, which was unfortunate. Whenever they got another task that wasn't as detailed, they got a bit helpless." - Technical architect

One can see that there are different opinions on task uncertainty. This may be connected to the roles and how they experience task uncertainty, which could be interesting to investigate further. However the technical architects had a good overview of the project progress, and the architect's statement suggests that it might be an increased task uncertainty among the teams during Phase 2, as requirements were less detailed. In general, tasks could be found in Jira, and any issues related to task uncertainty was easily solved by coordination mechanisms found in personal mode and group mode. Thus, task uncertainty was not considered a big issue for the investigated case. It showed however an increase in both personal and group coordination mode.

Task interdependencies

There was a larger degree of task interdependencies during Phase 1. Each team had a very large and detailed set of tasks to work on, that should be finished within the set sprint. This was found to be a problem during Phase 1, as the dependencies among teams caused delays and could push tasks into new sprints. Dependencies among teams were presented during planning meetings, and could also be found on Jira.

"In Phase 1 there were dependencies among the teams because you were dependent on user stories being finished. But at the start of every new sprint we got an overview over which user stories had dependencies to each other. And if there were user stories that caused dependencies across teams, we had to prioritize the stories that other teams were dependent on." - Front end developer 2

After changing to continuous deliveries in Phase 2, the amount of dependencies decreased, but it was now the teams' responsibility to keep track of dependencies and create relations across the teams.

The tasks also became smaller in Phase 2, which contributed to the decrease in dependencies. Functional architects were encouraged to cut down on user stories, which they managed, but they did encounter some problems with this in regards to technical dependencies.

"Yes, we managed to cut down on user stories. They almost disappeared, and it became a more ad hoc way to describe user stories. But I think the developers still worked by the water-scrum-fall process. It can still be difficult to break down user stories into smaller stories, while there are still technical dependencies. So we deliver large batches with stories, but there are 5 Jira-issues instead. I think it happens quite frequently. But I see a big difference in the teams that has gotten new functionality and the teams that works with old functionality. When we created the new team structure, two main projects were also created, "improvements" and "common functionality". And "common functionality" they got new products and could work in an entirely different way. They started with a concept and could have smaller user stories and deliver iterative and get feedback, while we had this old huge part to consider, and it was put together in such a way that creating a smaller story was challenging, according to a tech lead." - Product owner

Technical dependencies were still a problem, but this was mostly related to the older parts of the system, and was difficult to avoid.

In general, dividing the project into smaller pieces decreased the dependencies among the teams.

"The need for coordination did not disappear, but... I have an impression of that the arenas you used for communication disappears a bit. Maybe the need for it disappeared too. We tried to split the project in smaller pieces too, so every team got responsible for their own piece, which decrease the need for coordination." - Technical architect

"It used to be a lot of dependencies. There are less now." - Product owner

However difficulties were found in regards to prioritizations. In Phase 2, each team decided how they wanted to work and prioritize tasks, and this could affect dependencies to other teams:

"The problem now is that, you've got a team that has a lot to do, and then we have dependencies to what they are doing. Our tasks may not be a priority to them. That is something we struggle with daily" - Technical architect

The most important coordination modes in regards to task interdependencies for Phase 1 was found to be impersonal mode and group mode. Dependencies could be found on Jira and were always available to the teams. As well as group mode of coordination, due to the planning meeting where all the dependencies were presented.

For Phase 2, there were less dependencies, and no longer any planning meetings where all the teams were present. Group mode decreased due to this, but it was still important for functional roles to have meetings to discuss dependencies. It was also found that whenever teams had to collaborate or resolve problems, they would mostly use personal coordination mode, and communication were influenced by informal ad-hoc conversations.

Work unit size

During Phase 1, the teams usually consisted of developers, a Scrum master, and a tester. At this time the teams did not have all domain knowledge within the teams, as functional architects and technical architects acted more as associated members of the teams and did not spend much time with the teams on a daily basis.

Team size increased in Phase 2. As the teams became cross-functional and autonomous, functional roles and technical architects became more present and got a more fixed position within the teams. This was to ensure that all teams had necessary domain knowledge within the teams, and could work independently without being to dependent on people outside the teams.

When asked whether team size changed during the project, one of the project managers answered:

"Yes, it changed a bit. We used to have 7-8 people per team. We had a group of people who had the role 'functional architect'. They were responsible for the solution description. But each and one of them had a spot within a team. So they were kind of associated members of a team. They spent many of their weeks to be in meetings and speak to others with the same role and interacted with the customer. They did not only deliver a document to Confluence, they brought their mind into their team. To own it. So we had something corresponding for the testers. We had one tester per team. Which also had their own test community. A kind of virtual test team. But included them, we were about 9 people per team. And then the team size increased when we created the cross-functional teams." - Project manager

"Pretty flexible team sizes. It was probably some variations among the teams, but we were 8 people. Maybe not that many. We were quite a small team, and at some point we were 16-17." - Front end developer

The increase of team size did not seem to have a positive effect on the teams in terms of efficiency and communication within the teams, and it was difficult for people to keep track on a detailed level what other people were working on. One could see what tasks other people were assigned to on Jira, but they did not know the details of their work.

"The problem with large teams is that it is pretty hard to get an overview on what people are working on. You've only got a certain amount of capacity left to try to understand what people are doing. So the more people there are in a team, the more it gets like "ok, those two over there are doing something, but I don't know what, but that's ok". You get a division within the team. Or not a division, but more like a "silo of sources", which it is called. There is not a huge understanding on what people are doing, and you also loose some of that sense of group cohesion. So that is the problem with larger teams. I think you can do well with quite large teams, but you need to have clear tasks. Thing can escalate quite quick,

and maybe one will see that it is better to split into two or three teams, if you notice there are too little communication among the groups within the team." - Front end developer

"A functioning team should have a good atmosphere, socially or just that communication feels fine, and that often is a result of people working closely together and have something in common. And when you no longer have that, you loose a bit contact." - Front end developer

"There are more cooperation within a smaller team. Because you usually work with the same tasks. When you are a larger team, there are more tasks, and the tasks may be more diverse. So you end up working with different thing within the team." - Front end developer 2

"Yes, you've got Jira boards. You can see what people are working on, but it doesn't tell you that much. Even if you can see that they are working on something, I don't know what it is or what need that task cover, and it has got nothing to do with me. So you know what they are working on, but at the same time you don't." - Front end developer

"I prefer to work in smaller teams, 4-5 people. I think that is more effective. You can work close with each other, and the person next to you always know what you are working on. That will not happen in a team of 10 people." - Technical architect

It was also found that teams were rotated once in a while and people could switch to new teams.

"Yes, we rotated about 1-2 times a year maybe. Not everyone changed teams, but we did experiment to see who worked well together. It was depended on what knowledge they had. Sometimes a team perhaps needed more people on front end." - Technical architect

This caused that the barriers to go and talk to people on other teams were lowered whenever people knew each other. This caused the use of personal coordination mode to increase. Rotation on team members among the teams would therefore have a positive effect on communication and enable personal coordination mode.

"People switching teams did not cause any big changes, except that it could improve communication. It is much easier to go and speak to someone you used to be on a team with, rather than someone you have never spoken with. But as time went by you get to know pretty much everyone. So communication improved over time. Easier to talk to people, and you also know more about the product and domain." - Front end developer

"I think it helped that... since teams were changed in Phase 2, but we knew a lot of people from Phase 1, so we knew who to ask whenever issues arose. We knew each other, and that makes it easier to cooperate." - Front end developer 2

6 Discussion

This section discuss the findings presented in section 5, with the aim to reach a conclusion for the stated research question in section 1.2. The results are discussed in regards to the theoretical framework by Ven et al. (1976) and scientific literature on the topic of large-scale agile software development. This section will also provide limitations for this study.

6.1 Impersonal mode

With the impersonal mode of coordination, the findings showed that most of the same coordination mechanisms were present during both phases, however it was being used less in the second phase.

The methodology used in Phase 1 was a "water-scrum-fall"-methodology. It was said to be agile as they worked in sprints and used roles, events and artifacts from the Scrum methodology. However the great use of impersonal coordination mechanisms does not fit the agile principles (Beck et al., 2001). The investigated project relied heavily on detailed planning and documentation during the first phase. This made it difficult for the teams to work as agile as they would like, as they were restrained by detailed plans and requirements. Jira and Confluence were important tools for the impersonal mode, as plans, documentation and dependencies could be found here.

Working area was found to be an important coordination mechanism throughout both Phase 1 and 2. The teams in the investigated project were co-located within the same building, and mostly at the same floor, but there were times when some of the teams were placed at a different floor. This made the communication among the teams less efficient, and hindered coordination. Co-location worked as an enabler for personal coordination mode and group mode, and was seen as a necessity for successful coordination. The use of physical whiteboards was also used for coordination, and co-location would be a prerequisite for this coordination mechanism to work.

Being co-located in agile software development projects is also encouraged by Beck (1999), and the findings by Dingsøyr and Lindsjørn (2013), who indicates that co-location foster team performance. Research done by Strode et al. (2012) also considers co-location as an important factor for agile software development. However these studies only consider how being co-located affects a team within itself. The findings from the case study conducted in this thesis suggests that co-location is also highly important for coordination among teams in a multi-team environment.

In Phase 2 the same coordination mechanisms from Phase 1 were still used, but with one more added coordination mechanism: GitHub. With more impersonal coordination mechanisms present in Phase 2, it could seem that this may cause an increase in the impersonal mode, but the results showed the opposite. How the coordination mechanisms were used is an important factor to consider here. As the teams became autonomous with continuous deliveries, detailed planning and documentation decreased significantly in Phase 2. This caused Jira and Confluence to be less used as coordination mechanisms. Thus, the use of impersonal coordination mechanisms has decreased as a result of changing agile working methodology.

6.2 Personal mode

Personal coordination mode was found to increase in use from Phase 1 to Phase 2. Personal coordination mechanisms were used by teams who had dependencies to each other or to solve issues that suddenly arose. Instant messaging was frequently used as it was an easy and informal way to get in touch with people outside your own team, however this was mostly used to discuss smaller technical issues. To discuss problems related to dependencies, communication among teams typically went as one-on-one conversations through the Scrum masters during Phase 1. As the communication was influenced by the hierarchy it can be characterized as vertical communication.

During Phase 2 the use of personal coordination mode increased significantly. Due to the decrease in hierarchy, communication no longer went through Scrum masters. This lowered to barrier for the teams to reach out to each other and caused the communication to become mostly horizontal. Communication among the teams was also found to be more efficient due to this.

Kraut and Streeter (1995) also puts an emphasis on the importance of informal communication. They states that formal communication may not be sufficient to deal with uncertainty, and that informal communication is valuable for dealing with task uncertainty. The findings from Kraut and Streeter (1995) also shows that discussions with peers were found more valuable than group meetings or discussions with a boss. This is consistent with the findings in this thesis. Informal communication was important throughout the whole project, but especially for Phase 2, as there was a decrease in formal communication arenas such as scheduled meetings.

Another aspect that may have caused an increase in personal coordination mode is trust. This was stated by interviewees in this case study. Interviewees mentioned that it was much easier to talk to people they already knew. Trust increased throughout the project, as people got to know each other more. The importance of trust for coordination is substantiated by Osifo (2013) who has studied coordination, and summarizes findings from multiple studies related to coordination and trust. Trust can be seen as a factor for success, while the lack of it can cause failure (Smith & Schwegler, cited in Osifo (2013)). As personal coordination mechanisms is considered to be the most important for Phase 2, trust can be considered to be one of the factors contributing to successful use of personal coordination mechanisms.

As mentioned in section 6.1, co-location was also found to be a necessary contributing factor for the use of personal coordination mode. Without the teams being co-located the personal coordination mode would be much less prominent. The change of agile methodology had a clear effect on the large increase of personal coordination mode, as personal coordination mode was found to be more important than both group mode and impersonal mode for Phase 2.

6.3 Group mode

Group meetings were found to be the most used coordination mechanisms for Phase 1. Phase 1 used scheduled meetings that came with the use of the "water-scrum-fall" methodology. As the methodology changed, most the scheduled meetings were dropped. During Phase 2 it was found, however, that having some meetings were a necessity. Functional roles were often responsible for coordinating tasks and discussing dependency issues across the teams. Due to this, functional roles saw the need to re-introduce their meetings. In addition there was an increase in unscheduled meetings. A study by Dingsøyr et al. (2018) also found an increase in unscheduled meetings over time, as needs change over time. This is supported by Jarzabkowski et al. (2012), who also states that coordination mechanisms are dynamic practices. In addition, horizontal informal communication is found more useful in uncertain situations, rather than following hierarchy and rules (Argote, Ching et al., Crowston, as cited in Jarzabkowski et al. (2012)). The increase in task uncertainty for Phase 2 may therefore be a factor for the increased use of ad hoc meetings.

The findings from this case study suggests that it is necessary to use meetings as an arena for discussing dependencies among teams, and that relying on impersonal and personal coordination mechanisms alone is not sufficient. The number of group coordination mechanisms decreased for Phase 2, but the use of meetings were still necessary and scheduled meetings were rather replaced with ad hoc meetings whenever needed.

6.4 Comparison with findings by Dietrich et al. (2013)

Dietrich et al. (2013) have investigated inter-team coordination in multi-team projects, where case F can be considered to be the most similar to the case investigated in this thesis. We do not know the working methodology of the investigated case by Dietrich et al. (2013), but the number of people involved and the complexity of the project draws similarities to Phase 1 in the investigated project used for this thesis. This may indicate a correlation between the project size and -structure, and the use of coordination modes. The findings by Dietrich et al. (2013) showed that the use of impersonal coordination mechanisms and personal coordination mechanisms were found to be used at the same amount, while group mode was the most used.

In this thesis the use of group mode was also found to be the most important for Phase 1. Group coordination mechanisms, with focus on scheduled meetings, were frequently used throughout Phase 1 and played an important role for coordination. The use of impersonal coordination mechanisms and personal coordination mechanisms were both found to be important for Phase 1. The use of impersonal coordination mechanisms such as planning and documentation through Jira and Confluence were necessary at the beginning of the project and at the beginning of every sprint. Issues that occurred in the middle of a sprint, were mostly handled by personal coordination mechanisms. While impersonal mode was most used in the beginning of each sprint, personal coordination mode was used on a daily basis. Therefore, one can consider personal mode to be more important than impersonal mode for this phase.

6.5 Comparison with the hypotheses by Ven et al. (1976)

Ven et al. (1976) proposed three hypotheses on how task uncertainty, task interdependence and unit size may affect coordination modes, as presented in section 2.6.4. For task uncertainty, Ven et al. (1976) stated that an increase in task uncertainty is associated with a lower use of impersonal mode, a greater use of personal mode and a significantly greater use of group mode. The findings from the case used in this thesis, indicates that an increase in task uncertainty caused a lower use of impersonal mode, with a significant increase in personal mode. For group mode, the number of group coordination mechanisms decreased, but it was found an increase in ad hoc meetings. Based on Ven et al. (1976) an increase in task interdependencies is expected to cause a small increase in impersonal mode, moderate increase in personal mode, and a large increase in group mode. For this case, it was found to be a larger degree of task interdependencies in Phase 1. Group coordination mechanisms were frequently used for coordination related to task interdependencies, while personal coordination mechanisms were used to solve issues as they appeared. Task descriptions and overview over dependencies among the teams could be found on Confluence and Jira. As a result, impersonal mode was also used, but the findings indicates that the use of group mode and personal mode were more significant. For increase in work unit size, Ven et al. (1976) expected a decrease in group mode, an increase in personal mode, and a significant increase in impersonal mode. This is not consistent with the findings from this study. Phase 2 had an increased work unit size, but it was not found any increase in impersonal coordination mechanisms. It was however found a significant increase in personal mode and the use of ad hoc meetings within group mode.

6.6 Limitations

This thesis has used a theoretical model by Ven et al. (1976) as a basis for the analysis. This was argued to be a fitting choice for analyzing the case study, and worked well for its intended purpose. However, the use of only one model will only show results given from this point of view. Using multiple other models as a basis for the analysis would give a broader perspective and one could compare the results from using the different models, which could strengthen or weaken the findings.

As large-scale agile development is still a new research field, and particularly with focus on inter-team coordination, there is not found to be any research that study a case similar to the one used for this thesis. There were found some similarities between Phase 1 for this case study and findings by Dietrich et al. (2013) and (Dingsøyr et al., 2018), but it was found no studies on inter-team coordination with a similar methodology as used for Phase 2. This made it challenging to find related studies that could substantiate the findings.

7 Conclusion

The aim of this thesis has been to answer the research question "How is inter-team coordination affected by the change of agile methodology?". First, the topic of large-scale agile software development and coordination theory was investigated through a literature study. Through a case study on a large-scale agile project it was identified how inter-team coordination is affected by the change of working methodology. The results were analyzed by using a theoretical model on coordination modes by Ven et al. (1976). The model describes three different coordination modes, that each coordination mechanism can fit into: impersonal mode, personal mode and group mode. The results were also discussed in regards to related scientific literature.

7.1 What was found?

The investigated case was a large-scale agile project within a public Scandinavian organization. The project started out by using a "water-scrum-fall" methodology, and changed working methodology along the way. They dropped all Scrum roles, artifacts and meetings, and began to work more agile with cross-functional, autonomous teams and continuous deliveries. Phase 1 is used as term to describe the period before the change in working methodology, while Phase 2 is used as term to describe the period after the change.

It was found that for Phase 1 the most used coordination modes, in prioritized order, were group mode, personal mode, and impersonal mode. For Phase 2, the most used coordination modes in prioritized order were personal mode, group mode and impersonal mode, as illustrated in Table 15.

Table 15: Most used coordination modes for Phase 1 and Phase 2

Phase 1	Phase 2
1. Group mode 2. Personal mode 3. Impersonal mode	1. Personal mode 2. Group mode 3. Impersonal mode

7.2 How is inter-team coordination affected by the change of agile methodology?

The change in agile methodology caused a great increase in Personal coordination mechanisms. Although most scheduled meetings disappeared in Phase 2, the use of ad hoc meetings increased. The lowered use in impersonal coordination mechanisms, forced teams to collaborate more and it was necessary to communicate with each other across the teams. The communication become more horizontal and was more effective, as there was no longer any need to communicate through a Scrum master. Co-location was found to be important for Phase 1, but even more important for Phase 2, as it works as an enabler for personal mode and group mode.

As a conclusion to the stated research question one could see that the change of agile methodology caused a decrease in impersonal coordination mode, while group mode of coordination became less prominent and relied more on ad hoc meetings rather than scheduled meetings. Personal coordination mode experienced a large increase, which required that the coordination mode is enabled by the use of co-location, and less hierarchy to increase the horizontal communication.

7.3 What is the contribution of this thesis?

This thesis has investigated a large-scale agile software development project, that has changed working methodology from a "water-scrum-fall" methodology to high agility, autonomous and cross functional teams with continuous deliveries. It has not been found any other research that has studied the coordination mechanisms within a large-scale agile project that has gone through such a transition. Thus, this thesis provides an increased understanding on how inter-team coordination changes over time as a result of changes in agile practices. In addition the use of the theoretical framework by Ven et al. (1976) has not previously been used for comparing the change in use of coordination mechanisms. This thesis uses this framework to identify these changes, and points out how the change in agile methodology affects coordination mechanisms. This thesis could be found interesting for project managers considering to proceed with such a transition in their project. As inter-team coordination is considered to be a prominent challenge for multi-team projects, it is interesting for project managers to look at what coordination mechanisms are present when a large-scale agile project succeeds with such a transition.

Students and researchers within the field could also find this thesis useful. The findings creates a basis for further research, and there was discovered a lack of research on the field of inter-team coordination for large-scale scale agile development projects that could substantiate the findings of this thesis. Suggestions of further research will be provided in section 7.4.

7.4 Future work

Based on the findings from this thesis, it is recommended to conduct further research on the field of large-scale agile development with focus on inter-team coordination. The findings indicates that there were differences in how the roles coordinated with each other. It would therefore be interesting to not only look at teams as an entity, but also focus on how different roles among the teams coordinate with each other.

More case studies on how inter-team coordination is affected by change of agile methodology should also be conducted, as there was found a lack of research on this topic. In particular, studies on how coordination is affected by the use of autonomous, cross-functional teams with continuous deliveries should be done. Multiple studies can help substantiate each other and increase the validity. Conducting a similar case study with the use of different coordination theories would also provide more perspective on the topic.

Bibliography

Beck, K., 1999. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D., 2001. *Manifesto for Agile Software Development.* Retrieved October 16, 2019, from `http://www.agilemanifesto.org/`.

Bell, T.E., Thayer, T.A., 1976. *Software Requirements: Are They Really a Problem?* ICSE '76 Proceedings of the 2nd international conference on Software engineering , 61–68.

Boos, M., Kolbe, M., Kappeler, P., Ellwart, T., 2011. Coordination in Human and Primate Groups. Springer-Verlag, Berlin, Heidelberg. doi:`10.1007/978-3-642-15355-6`.

Brooks, F.P.J., 1975. *The Mythical Man-Moth.* Addison-Wesley.

Cho, J., 2008. *Issues and Challenges of Agile Software Development With Scrum.* Issues in Information Systems .

Conboy, K., Carroll, N., 2019. Implementing large-scale agile frameworks: Challenges and recommendations. IEEE Software , 44–50doi:`10.1109/MS.2018.2884865`.

Crowston, K., 1994. A Taxonomy of Organizational Dependencies and Coordination Mechanisms. Working Paper Series. MIT Center for Coordination Science.

Dictionary.cambridge.org, 2019. *SCRUM: meaning in the Cambridge English Dictionary.* Retrieved November 11th, 2019, from `https://dictionary.cambridge.org/dictionary/english/scrum`.

Dietrich, P., Kujala, J., Artto, K., 2013. Inter-team coordination patterns and outcomes in multi-team projects. Project Management Journal 44. doi:`10.1002/pmj.21377`.

Dikert, K., Paasivaara, M., Lassenius, C., 2016. Challenges and success factors for large-scale agile transformations: A systematic literature review. Journal of Systems and Software 119, 87–108. doi:10.1016/j.jss.2016.06.013.

Dingsøyr, T., Falessi, D., Power, K., 2019. Agile development at scale: The next frontier. IEEE Software 36, 30–38. doi:10.1109/MS.2018.2884884.

Dingsøyr, T., Fægri, T., Itkonen, J., 2014. What is large in large-scale? a taxonomy of scale for agile software development. doi:10.1007/978-3-319-13835-0_20.

Dingsøyr, T., Lindsjørn, Y., 2013. Team performance in agile development teams: Findings from 18 focus groups , 46–60.

Dingsøyr, T., Moe, N., 2014. Towards principles of large-scale agile development: A summary of the workshop at xp2014 and a revised research agenda. doi:10.1007/978-3-319-14358-3_1.

Dingsøyr, T., Moe, N.B., Seim, E.A., 2018. Coordinating knowledge work in multiteam programs: Findings from a large-scale agile development program. Project Management Journal 49, 64–77. doi:10.1177/8756972818798980.

Dybå, T., Dingsøyr, T., 2009. What do we know about agile software development? Software, IEEE 26, 6 – 9. doi:10.1109/MS.2009.145.

Gittell, J., 2006. Relational coordination: Coordinating work through relationships of shared goals, shared knowledge and mutual respect. Relational Perspectives in Organizational Studies: A Research Companion , 74–94.

Highsmith, J., 2002. *Agile Software Development Ecosystems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Hoda, R., Noble, J., Marshall, S., 2012. Developing a grounded theory to explain the practices of self-organizing agile teams. Empirical Softw. Engg. 17, 609–639. doi:10.1007/s10664-011-9161-0.

International Telecommunication Union, W.T.D.R., database, 2019. Individuals using the internet (% of population). Retrieved October 16, 2019, from https://data.worldbank.org/indicator/it.net.user.zs?end=2005&start=1990&view=chart.

J. Highsmith, K. Orr, A.C., 2000. *Extreme programming.* E-Business Application Delivery , 4–17.

Jarzabkowski, P.A., Lê, J.K., Feldman, M.S., 2012. Toward a theory of coordinating: Creating coordinating mechanisms in practice. Organization Science 23, 907–927. doi:10.1287/orsc.1110.0693.

Jørgensen, M., 2019. Relationships between project size, agile practices, and successful software development: Results and analysis. IEEE Software 36, 39–43. doi:10.1109/MS.2018.2884863.

Kraut, R.E., Streeter, L.A., 1995. Coordination in software development. Commun. ACM 38, 69–81. URL: https://doi.org/10.1145/203330.203345, doi:10.1145/203330.203345.

Leffingwell, D., 2011. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. 1st ed., Addison-Wesley Professional.

Litwak, E., Hylton, L.F., 1962. Interorganizational analysis: A hypothesis on co-ordinating agencies. Administrative Science Quarterly 6, 395–420.

Malone, T., 1988. What is coordination theory? Massachusetts Institute of Technology (MIT), Sloan School of Management, Working papers .

Malone, T., Crowston, K., Lee, J., Pentland, B., Dellarocas, C., Wyner, G., Quimby, J., Osborn, C., Bernstein, A., Herman, G., Klein, M., O'Donnell, E., 1999. Tools for inventing organizations: Toward a handbook of organizational processes. Former Departments, Centers, Institutes and Projects 45. doi:10.1287/mnsc.45.3.425.

Malone, T.W., Malone, T.W., Crowston, K., 1994. The interdisciplinary study of coordination. ACM Comput. Surv. 26, 87–119. URL: http://doi.acm.org/10.1145/174666.174668, doi:10.1145/174666.174668.

Merriam-Webster, 2019. *Dependency.* Retrieved December 6th, 2019, from https://www.merriam-webster.com/dictionary/dependency.

van Mierlo, H., Rutte, C.G., Vermunt, J.K., Kompier, M.A.J., Doorewaard, J.A.M.C., 2006. Individual autonomy in work teams: The role of team autonomy, self-efficacy, and social support. European Journal of Work and Organizational Psychology 15, 281–299. doi:10.1080/13594320500412249.

Moe, N., Dingsøyr, T., Dybå, T., 2010. Overcoming barriers to self-management in software teams. Software, IEEE 26, 20 – 26. doi:10.1109/MS.2009.182.

Oates, B.J., 2006. Researching Information Systems and Computing. Sage Publications Ltd.

Osifo, C., 2013. The effects of coordination on organizational performance: An intra and inter perspective. Asian Journal of Business and Management 01.

Reifer, D., Maurer, F., Erdogmus, H., 2003. Scaling agile methods. Software, IEEE 20, 12 – 14. doi:10.1109/MS.2003.1207448.

Rolland, K., Fitzgerald, B., Dingsøyr, T., Stol, K.J., 2016. Problematizing agile in the large: Alternative assumptions for large-scale agile development completed research paper doi:10.13140/RG.2.2.27795.07207.

Royce, W.W., 1970. *Managing the Development of Large Software Systems.* The proceedings of the WESCON , 328–339.

Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14, 131–164. doi:10.1007/s10664-008-9102-8.

Schwaber, K., Beedle, M., 2001. *Agile Software Development with Scrum.* 1st ed., Prentice Hall PTR, Upper Saddle River, NJ, USA.

Sommerville, I., 2011. *Software Engineering.* 9 ed., Addison-Wesley.

Stavru, S., 2014. A critical examination of recent industrial surveys on agile method usage. Journal of Systems and Software 94, 87 – 97. doi:`https://doi.org/10.1016/j.jss.2014.03.041`.

Stray, V., Moe, N., Hoda, R., 2018a. Autonomous agile teams: Challenges and future directions for research. doi:`10.1145/3234152.3234182`.

Stray, V., Moe, N., Hoda, R., 2018b. Autonomous agile teams: Challenges and future directions for research doi:`10.1145/3234152.3234182`.

Stray, V., Moe, N.B., Aasheim, A., 2019. Dependency management in large-scale agile: A case study of devops teams .

Strode, D., Huff, S., Hope, B., Link, S., 2012. Coordination in co-located agile software development projects. Journal of Systems and Software 85, 1222–1238. doi:`10.1016/j.jss.2012.02.017`.

Strode, D.E., 2016. A dependency taxonomy for agile software development projects. Information Systems Frontiers 18, 23–46. doi:`10.1007/s10796-015-9574-1`.

Sutherland, J., 2001a. Agile can scale: Inventing and reinventing scrum in five companies 14, 5–11.

Sutherland, J., 2001b. *Inventing and Reinventing SCRUM in Five Companies* .

Sutherland, J., Schwaber, K., 2017. *The Scrum Guide.* Retrieved October 22, 2019, from `https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf`.

Sutherland, J., Scrum Inc., 2019. *The Scrum At Scale® Guide.* Retrieved November 13, 2019, from `https://www.scrumatscale.com/scrum-at-scale-guide-read-online/`.

Takeuchi, H., Nonaka, I., 1986. *The New New Product Development Game*. Harvard Business Review .

Thompson, J.D., 1967. Organizations in action; social science bases of administrative theory. McGraw-Hill.

Ven, A., Delbecq, A., Koenig, J., 1976. Determinants of coordination modes within organizations. American Sociological Review 41. doi:10.2307/2094477.

VersionOne, C., 2019. *The 13th Annual State of Agile Report*.

Viechnicki, P., Kelkar, M., 2017. *Agile by the numbers*. Retrieved October 18, 2019, from https://www2.deloitte.com/us/en/insights/industry/public-sector/agile-in-government-by-the-numbers.html.

Vijayasarathy, L., Butler, C., 2015. *Choice of Software Development Methodologies - Do Project, Team and Organizational Characteristics Matter?* IEEE Software 1, 86–94. doi:10.1109/MS.2015.26.

Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering URL: https://doi.org/10.1145/2601248.2601268, doi:10.1145/2601248.2601268.

Yin, R.K., 2009. Case study research: design and methods. Sage.

© Atlassian, Inc., 2020a. Confluence: Features functions. Retrieved March 13th, 2020, from https://confluence.atlassian.com/confeval/confluence-evaluator-resources/confluence-features-functions.

© Atlassian, Inc., 2020b. Jira | issue project tracking software | atlassian. Retrieved March 13th, 2020, from https://www.atlassian.com/software/jira.

© Scaled Agile, Inc., 2019a. *Core Values - Scaled Agile Framework.* Retrieved November 11, 2019, from
`https://v46.scaledagileframework.com/safe-core-values/`.

© Scaled Agile, Inc., 2019b. *Scaled Agile Framework - SAFe for Lean Enterprises.* Retrieved November 11, 2019, from
`https://v46.scaledagileframework.com/`.

Appendix A

Interview guide: An investigation on large-scale agile software development

Inter-team coordination

Project beginning - Context:

- Project elaboration
- Anything different compared to other projects you've participated in?
- Physical environment: Office workspace, whiteboards, meeting locations
- Comparison with other projects

Your role:

- What was your role, and what did you do?
- Responsibilities?
- Have you had this role or similar roles before? What was it like here compared to other experiences?
- Challenges?
- With your role, who else do you need to work closely with or take into account?
- Has it been any changes so far? Any larger events that influenced the way you worked?
- How did you experience the work within your team?

Team coordination:

- In which contexts were coordination among teams necessary?
  Solutions description - construction - test

- Which dependencies do you have to other teams? (examples?)

  Dependencies among requirements / technical dependencies?

- How did you manage the dependencies?

- Which arenas did you use for team coordination?

- How do you think the inter-team coordination worked out?

- Changes that happen through time? (from formal to informal)

- Changes in how the teams were organized - How did you experience this?

- Coordination towards other participants?

- Dependencies to other projects/systems/processes within the customer organization.

- Examples on how you solved dependencies?

- What is important to handle such dependencies?

Appendix B

Interview guide

February 2020

Introduction:

- What was your role in the project?
- When did you became a part of the project?

Dependencies:

- What was the dependencies among the teams like throughout the different project phases? Examples?
- Did you keep track of dependencies? How?
- Who was responsible for making sure that responsibilities were solved?
- How did you solve dependencies? Which arenas did you use?
- Did new arenas occur during the last phase? Did you re-introduce any arenas that were used earlier in the project?
- Could the dependencies be solved in another way? Examples?
- Is there any way of coordinating that you think worked particularly well or did not work? Why?
- Was there any dependencies that you were not able to predict?
- What was the use of documentation and planning like during the first phase, versus the last phase?

Communication:

- How did the teams communicate throughout the different phases? Was there any changes?
- How was the use of Slack? Was there any other tools that were used for communication? What type of communication were the different tools used for?
- How was the use of group meetings? What kind of scheduled meetings were used? How often were they held?
- How was the use of unscheduled meetings?
- Did the use of meetings change throughout the project? How was it during the first phase, versus the last phase?
- Who participated in the different meetings?
- Were there any meetings where all the teams were gathered?
- Whenever only one person from the team attended meetings, what type of role did they have?
- What do you think of the use of group meetings in general, and how did it affect coordination?
- Did you ever experience uncertainty regarding tasks, how they should be done or prioritized? What did you do if you experienced such uncertainty?

Teams:

- How was the team sizes during the first phase, versus the last phase? Did the team size change?
- Did people get replaced a lot? If so, was this new people from outside the project, or did people switch among the teams? How did this affect the team?
- Was there any changes in roles within the teams?

- Whenever team size increased, how did this affect the team?
- How did office workspace and location affect coordination?

Final thoughts

- Is there anything else we should discuss? Anything you would like to add in regards to coordination?

Camilla Tøftum Ranner

Large-scale Agile Software Development