NTNU
Norwegian University of
Science and Technology

Tailoring Agile Methods for Large Projects

A Case Study of a Large Agile Project

Stein-Otto Svorstøl

Master of Science in Computer Science
Submission date: August 2017
Supervisor: Torgeir Dingsøyr, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

The adoption of agile methodologies is increasing. The methods are usually designed for one team of 5-12 people. This makes the adoption challenging for large organizations. Little research has been conducted on the implementation of agile methods in large projects. Therefore this master thesis describes a large project using agile methods.

To describe the project there were conducted 21 interviews with people working on it. The analysis of the interviews lead to a description of how the project has been organized, and how they have tailored the Large-Scale Scrum (LeSS) Framework to their context. The tailorings include removing activities with both teams and introducing an advisory meeting where business developers could get advice from architects on their proposed features. There were also done tailorings to enable using the method in an enterprise context, such as status meetings and a Project Management Office which in turn was controlled by line managers. These tailorings seem to be linked to the following two challenges:

- **Company organization and policy:** The line managers in the company demand reports and extra meetings that otherwise would not be necessary for an agile project. This is experienced as a both time consuming and limiting to the development, by the team members.
- **Understanding of agile methodologies in the company:** Business developers and others not working with software development are unfamiliar with agile methodologies, which also seems to be a limiting factor, as the development process is not completely understood.

The challenges and the benefits of the implementations have been compared with other case studies, and similarities have been identified.

The findings can be used when implementing agile in other large organizations. It can also be used to rework existing agile methodologies to better suit large organizations and projects. This can transform current methods so that more organizations can make use of them.

Sammendrag

Bruken av smidig metodikk øker. Metodene er vanligvis laget for et team med 5-12 medlemmer. Dette gjør innføringen utfordrende for større organisasjoner. Det finnes lite forskning på implementasjonen av metodene i store organisasjoner. Denne masteroppgaven beskriver derfor bruken av smidige metoder i et stort prosjekt.

For å beskrive prosjektet ble det gjennomført 21 intervjuer med mennesker som jobbet på det. Analysen av intervjuene førte til en beskrivelse av hvordan det har blitt organisert, og hvordan de har tilpasset Large-Scale Scrum (LeSS)-rammeverket til deres kontekst. Tilpasningene var blant annet at de fjernet aktiviteter hvor flere team deltok, og at de innførte et rådgivningsmøte hvor forretningsutviklere kunne få råd fra arkitekter, angående deres foreslåtte funksjonalitet. Det ble også gjort tilpasninger for å sørge for at metoden kunne brukes i en bedriftssammenheng, for eksempel flere statusmøter og et Programkontor som styrte prosjektet. Dette ble igjen kontrollert av linjeledere i en kontrollgruppe. Disse tilpasningene later til å være knyttet til to utfordringer:

- **Bedriftens organisering og retningsliner:** Linjelederne i bedriften ønsker rapporter og ekstra møter som ellers ikke ville vært nødvendig i et smidig prosjekt. Dette er sett på som begrensende og noe som sluker tid av de som er involvert.
- **Forståelse for smidig metodikk innad i selskapet:** Forretningsutviklere og andre som ikke jobber med programvareutvikling er ikke så godt kjent med smidige metodikker, noe som også virker som en begrensende faktor, fordi utviklingsprosessen ikke er helt forstått.

Utfordringene og fordelene med implentasjonen er sammenlignet med andre case-studier, og likheter har blitt identifisert.
Funnene kan bli brukt når man skal implementere smidig i andre store organisajsoner. De kan også bli brukt til å omarbeide eksisterende smidige metodikker slik at de bedre passer store organisasjoner og prosjekter. Dette kan forandre eksisterende metoder slik at flere organisasjoner kan ta dem i bruk.

Acknowledgements

This master thesis was written at Norwegian University of Science and Technology (NTNU) during the spring of 2017, after writing a pre-master project during the fall of 2016. The pre-master project was a literature review, while this thesis is an empirical study. It is a result of several hours of interviews, interview transcriptions, reading interviews, analyzing interviews and of course writing this very thesis. It has been boring, exciting and interesting, all at the same time.

There are a few people who enabled me to write this thesis. First those that participated in the study, as well as those that helped me get in touch with the company I did the study on. Due to anonymisation, I do not wish to print their names, but they know who they are, and they all have my deepest gratitude. Thank you.

Next, my supervisor Torgeir Dingsøyr assisted me through helpful tips about conducting the research, as well as feedback on the thesis itself. Without his pointers, I would never manage to complete this thesis.

I also want to thank my girlfriend Anna Kastet for her advice and suggestions.

Finally, I want to thank *you* for reading this paper: Thank you.

Oslo, August 12, 2017

[illegible]

Stein-Otto Svorstøl

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Agile is a philosophy within the field of software engineering, which has sprung from what is called *The Agile Manifesto* released in 2001.(*The Agile Manifesto*) Based on that manifesto, new methodologies have been developed, such as Scrum and Extreme Programming (XP) (Sutherland, 2014; Beck and C., 2004). The interest in adopting agile methods has increased as well: The company Version One who annually conducts surveys on the usage of agile in the industry, found in the VersionOne, 2016, that the number of large organizations that respond to their survey is increasing, from 24% in 2015 to 26% in 2016. The so-called "agile sweet spot" is teams of 5-12 people. (Nord, Ozkaya, and Kruchten, 2014) How may one combine and tailor these agile methods for large corporations? Through a case study, this master thesis will describe how it might be done. It will also describe the benefits and challenges the organization has with agile methods, so that other businesses may learn from them.

The following chapter will first explain the research motivation for this thesis, as well as the author's personal motivation for going into this specific type of study. Then the chapter will introduce the research questions, give a brief introduction to the case through which the research questions will be explained, explain the scope of the study and describe the intended audience of the paper. Finally, this chapter will give an overview of the structure of the rest of this thesis.

1.1 Research Motivation

To motivate the need for research into agile in large projects, and how one can tailor the methods, this section will first look at agile methods and their advantages in Section 1.1.1. Then Section 1.1.2 will define what a *large* project is, and finally Section 1.1.3 will motivate for the need for agile methods in large projects.

1.1.1 Agile Methodologies and their Benefits

In the 1980s and early 90s, the dominant view within software engineering was that successful project planning leads to successful software and projects. This was not always the case, and in the late 90s, many grew tired of this way of working, which lead to a new philosophy to drive software development: Agile. (Sommerville, 2011, p. 58)

Agile methodologies are ways of working with software engineering that are based on the agile philosophy. (Dingsøyr et al., 2016) The philosophy was first defined in The Agile Manifesto released in 2001. (*The Agile Manifesto*) It reads:

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> - *Individuals and interactions over processes and tools.*
> - *Working software over comprehensive documentation.*
> - *Customer collaboration over contract negotiation.*
> - *Responding to change over following a plan.*
>
> *That is, while there is value in the items on the right, we value the items on the left more. (Sommerville, 2011, p. 59)*

The agile manifesto shows how these new methods differ from the old plan-based methods, such as waterfall. The authors meant that methods such as waterfall valued the process more than the products the process was intended to produce. The argument is that the plan-based methods spend more time producing documentation and plans, rather than developing a product. In other words: The Manifestos authors and the agile movement strive for methods that produce what the customer wants, with an appropriate amount of documentation. (Sommerville, 2011, p. 29)

Based on the agile manifesto there have been multiple modern agile methodology frameworks. The most famous one is Extreme Programming (XP), created by Kent Beck in 1999. (Sommerville, 2011, p. 59)(Beck and C., 2004) Later on, Sutherland's method Scrum became quite famous, and widely adopted in the industry. (Sutherland, 2014; VersionOne, 2016; Rodriguez et al., 2012; Dingsøyr et al., 2012) The "latest" agile methodology is the Lean Startup (LSU), which is said to be especially ideal for developing new concepts and products. (Ries, 2012) All these methodology frameworks vary in implementation, but are based on the ideas in the Agile Manifesto. These agile methods have been shown to be successful and beneficial. (Sommerville, 2011, p. 59) *VersionOne's 11th Annual State of Agile Report* shows that their respondents experience benefits of using agile. They also report on how many of the respondents experience various benefits. Their results are shown in Table 1.1. In addition, empirical research has also found these advantages. Petersen and Wohlin (2009) compared the advantages and issues of state of the art agile methods such as XP and Scrum with those that they identified in a case study, and found that "...the case study and literature agree on the benefits".

Table 1.1: Overview of "benefits of adopting agile" from VersionOne, 2016, and how many respondents to the survey experienced each benefit. Respondents could select multiple alternatives in the survey.

Benefit	**Experiencing the benefit**
Ability to manage changing priorities	88 %
Project visibility	83 %
Increased team productivity	83 %
Delivery speed/Time To Market (TTM)	88 %
Team Morale	81 %
Business/IT alignment	76 %
Software quality	75 %
Project predictability	75 %
Project risk reduction	74 %
Engineering discipline	68 %
Software maintainability	64 %
Managing distributed teams	61 %
Project cost reduction	56 %

The benefits can also be seen in Sommerville, 2011, p. 33, who states that there are three advantages of “incremental development . . . which are a fundamental part of agile approaches”, over the waterfall model, which traditionally has been used in large enterprises. His three advantages summarized what VersionOne found. The three advantages of incremental development stated by Sommerville are:

- Reduced costs and overhead
- It is easier to get customer feedback
- More rapid deliveries

The agile methods have been best suited for *small* teams that are building a *small* or medium-sized product for sale, or that are building custom software (Sommerville, 2011, p. 59-60). Agile frameworks are usually defined to be used in what is called the “sweet spot of agile”, which is defined as teams of 5-12 collocated developers. (Nord, Ozkaya, and Kruchten, 2014; Hobbs and Petit, 2017) An agile project should also have low to medium criticality. (Nord, Ozkaya, and Kruchten, 2014) The usage of agile outside the "sweet spot" is changing, due to factors explained in Section 1.1.3. First, there is a need to define what is meant by a *large project*. This is explained in the next section, Section 1.1.2.

1.1.2 What is a *Large* Project?

To be able to reason about the use of agile outside the “sweet spot of agile”, one must first have a clear definition of what this “outside” is: What is a *large* project?

A large project, or a *large-scale* project, is a loose term in itself. In 2014 there was held a workshop on large-scale *agile* projects, and among other things, the definition of large-scale was discussed. The various suggestions are found in the summary from the workshop. (Dingsoyr and Moe, 2014) The suggested definitions are wide-ranging, which is reflected in the various types of large-scale projects, which was also discussed at the workshop. There are large projects that are developed by huge corporations, and those that are developed by volunteers that communicate over the Internet. Another type of large-scale projects are those with distributed teams, e.g. outsourced teams that do testing. Some suggestions for definitions were number of lines of code, the size of the organization owning the project, the number of activities needed for coordination, how crucial the project was and the number of developers or teams in the project.

Because of all the various explanations of "large project," it is important to clarify what is meant by the term, in the context of this paper. Here, the terms "large project" or "large-scale project," mean *a project that has at least two teams.*

1.1.3 The Need for Agile in Large Projects

In Section 1.1.1 the benefits of agile were explained, and in Section 1.1.2 *large projects* were defined for this paper. While there exist large projects, one can wonder if they are using agile methods. There are at least some: This is shown in that it is not only the organizations that are large in the survey by VersionOne, 2016, but also the software organizations within the company. The number of people within the software organizations of the respondents of the VersionOne, 2016, are shown in Table 1.2. When there are so many large software development organizations, one must also assume that there exist many large projects. VersionOne, 2016 also found that:

> *. . . although 44% of respondents stated that they were extremely knowledgeable regarding agile development practices, 80% said their organization was at or below a "still maturing" level.*

Table 1.2: The percentage of respondents using agile methods, sorted on the the number of people within their development organizations, in the agile surveys of VersionOne in 2015 compared to 2016. (VersionOne, 2015; VersionOne, 2016)

Number of people	**Usage in 2015**	**Usage in 2016**
101-1000	31%	34%
1001-5000	15%	15%
5001+	16%	19%

This finding means that here exist many large organizations implementing agile and that agile methods are still maturing in these organizations. Given the "sweet spot" of 5-12 co-located developers there necessarily have to be multiple teams with development organizations of over 100 people. Table 1.2 also shows an increase in the adoption of agile in large development organizations from 2015 to 2016.

The reason for this increase in usage, is that the organizations also want the benefits of agile, explained in Section 1.1.1. The benefits might increase customer satisfaction by creating solutions that more directly cater customer needs. Rapid deliveries mean short TTM for new features, which can increase market shares. Reduced cost and overhead is of course also good for businesses developing systems, because it means they can save money, and use it on e.g. other projects. It may also be that large companies now *need* to adopt more agile methods, to be able to make competitive products, and meet customer demands before the customer moves to another provider. Highsmith and Cockburn wrote "Expectations have grown over the years. The market demands and expects innovative, high quality software that meets its needs — and soon.". The same has even been reported by news media, such as in the article *Innovate or die: The stark message for big business* from BBC News, which states: "Big business has to innovate or die". (Wall, 2014).

The government also has a need to innovate and create services that its citizens want. The Swedish author Söderström released a book called *Jævla drittsystem!* (Norwegian title, title translated to English is *Stupid bloody system!*) in 2013. In his book, he makes the point that traditional computer systems are not made on the user's terms and that the users should demand better systems. He gives examples from the Swedish government, but also from the private sector. The points are in other words valid for governmental systems, commercial system, and Business-to-Business IT-systems. The increased importance of IT in government can also be seen in something as simple as the name of one of the ministers in the Norwegian government, and his office. Until 2014 one of the ministries was called *Ministry of Local and Regional Government*, and now it is called *Ministry of Local Government and Modernisation*. Other than having "modernisation" in its name, it also has responsibility for departments such as *Agency for Public Management and eGovernment* and *Norwegian Data Protection Authority*. All this shows that both governments and business has to be able to create new IT-systems to be able to produce usable services today. When governments and large businesses create systems, the systems are usually large.

In other words: It is necessary that new systems are made, and in the context of some organizations, the challenges are necessarily so large that the projects become big. It is also necessary that the IT-systems are made on the user's terms. For this, agile methods are great, as described in Section 1.1.1. However, how enterprises can use agile methods in their large projects, has been a challenge for the industry for many years. (Dingsøyr et al., 2012; Dingsoyr and Moe, 2014) So how can an organization that is large, or a large project in itself, use agile methods? That is the topic of this thesis.

1.2 Personal Motivation

The author of this thesis wrote a literature review on the role of architecture in large-scale agile development during the fall of 2016. The work sparked an interest to look further into how agile methods are implemented. The reason for this was that the work made the author see how books on agile methods often seem to present them as a silver bullet, even though there are a variety of challenges associated with them. The author thought that showing and explaining these challenges would help the community to find solutions to them, which in turn will advance the field of software development methodologies, and in turn help creating great IT-systems.

1.3 Research Questions

In the previous section, the motivation for doing research into method-tailoring in large organizations was explained. Translated into a research question, it sounds like this:

> *How can a large organization use agile methodologies in a development project with multiple teams, so that the teams can develop systems that serve the needs of multiple stakeholders within the organization?*

To be able to answer the main research question there is a need to answer the following sub-questions:

RQ1: How is the development project organized in terms of teams, roles, meetings, and activities?

RQ2: How has the organization tailored a agile methodology, so that it fits the organizations enterprise needs?

The goal of this paper is to answer these questions.

1.4 The Case of Study

To be able to explore and answer the research questions presented in Section 1.3 there was a need for a case to study. Through contacts in the industry, the author found a large organization that was doing a multi-team development project. The company has around 3500 employees in the Scandinavian and Baltic countries. The development organization in which this study is conducted consists of around 200 people. The project was said to have at least two teams, and these teams interacted with business analysts other places in the organization. The complete development program was said to consist of around 50 people, and the development methodology was said to be inspired by the LeSS Framework. The LeSS Framework will be explored in Section 2.2. The author was provided access to the organization's building and IT-systems so that that research could be planned and conducted. How the case was selected is further described in Section 3.2.3.

1.5 Research Scope

This paper is the master thesis of the author Stein-Otto Svorstøl, done at NTNU during the Spring of 2017. The master thesis is worth 30 points in the European Credit Transfer and Accumulation System (ECTS). This is meant to be equivalent to 750-900 work hours. This includes planning and conducting the research, and writing the paper.

1.6 Intended Audience of the Paper

The intended audience for this paper, are those that are currently using agile methods in their large projects or are planning on using them. Also, those that develop agile enterprise methods, or scaled agile methods, may have use of this paper. These two groups can use the results to adjust their project methodology, or methodology framework, to better suit large projects.

The author advises that the reader has an understanding of *The Agile Manifesto*, and be familiar with agile methodology frameworks made for the agile sweet spot, such as Extreme Programming (XP), Scrum, the Lean Startup (LSU) and Kanban. (Beck and C., 2004; Sutherland, 2014; Ries, 2012; Sugimori et al., 1977; Ahmad, Markkula, and Oivo, 2013) Readers are also advised to familiarize themselves with Atlassian JIRA[1], which is used in the case and is also the most used tool for project management in the industry. (VersionOne, 2016, p. 16)

1.7 Overview of this Paper

To be able to answer the research questions given in Section 1.3, through the study of the case introduced in Section 1.4, there is a need to introduce the LeSS Framework, as it has inspired the methodology used in the case. There is also a need to look at similar studies, and describe existing models for handling architecture in such a project. This will be done in Chapter 2, and used when the results of the study are to be discussed. Then, Chapter 3 will explain how the research was conducted. Chapter 4 will show the results of the research, as well as discuss them in the light of background given in Chapter 2. Chapter 5 will provide a conclusion, and give a direction for future research that can be done based on this research.

[1]https://www.atlassian.com/software/jira

Chapter 2

Theory

Chapter 1 motivated for researching the area of large-scale agile, and briefly introduced the case. To be able to understand the findings of the case study, it is necessary to give some background information on the aspects of the findings of the case: Software development methodologies, the Large-Scale Scrum (LeSS) Framework, other cases of method tailoring and their results, and finally software architecture. This chapter will give the reader insight into these areas.

First, in Section 2.1, the concept of software development methodologies will be explained. Then, in Section 2.2, the Large-Scale Scrum (LeSS) Framework which the methodology in the case is based upon, is described. Section 2.1.2 will give an overview of other case studies, which the findings of this case study can be compared to. Finally, in Section 2.4 and Section 2.5, will provide some background for the role of architecture in large-scale development. All this will give the reader a foundation for understanding and reasoning about the findings, which will be presented in Chapter 4.

2.1 Software Development Methodologies

Before one can investigate the various aspects of tailoring agile methods, one must first define the term "software development methodology". One can first start with defining "software process", or "software development process". It can be defined in the following way:

> *A software process is a set of related activities that leads to the production of a software product. Sommerville, 2011, p. 59*

This means that the development process is simply a term for activities that produce the software product, from requirement elicitation to programming. Then, what are software development methodologies? The dictionary Merriam Webster[1] defines the word "methodology" as "a body of methods, rules, and postulates employed by a discipline: a particular procedure or set of procedures". In other words, a "software development methodology" is a specific procedure for the software development process. They have methods, rules, and postulates for developers to base their software development process upon. This word can then be used to describe *methodologies* such as Scrum. (Sutherland, 2014) These defined methodologies are also known as *methods* or *frameworks*. The idea is the same: They are a defined approach to the software process.

2.1.1 The Need for Methodologies

In the previous section, the term *software development methodology* was defined, and exemplified with Scrum. But why is there a need for methodologies?

According to Sommerville, the software development process has four activities which are "fundamental to software engineering". They are:

1. Software specification, where one defines the functionality and requirements of the system to be made.
2. Software design and implementation, where one designs the system and implements it.
3. Software validation, where one validates that the defined requirements are met.
4. Software evolution, where one changes the software with new needs, e.g. add new features or change existing ones.

[1]https://www.merriam-webster.com/

These four activities can be carried out in a variety of ways, and one can come up with one's own approaches to these fours steps. Why then, use an existing defined approach? Sommerville, 2011, p. 29 explains this by listing the advantages of a standardized software process: Selecting a standardized process will reduce diversity in the process in an organization, which in turn improves communication. It will also reduce training time, as one is already familiar with the process. Also, using a standardized software development process can allow automation of process support.

In short: To use a process that is proven to work well in other projects, can reduce the training time and improve communication. This will reduce cost. Choosing standardized processes will also reduce cost because one does not have to develop the process from scratch.

Software development methodologies are "standardized processes" as described by Sommerville: They provide a set of roles and activities that one can use to produce a software product. Often the standardized processes are called frameworks.

2.1.2 Tailoring Methodologies

Section 2.1.1 described the advantages of standardized software development processes. One may experience that the selected methodology has limitation within the context it is to be used. Sommerville wrote that "There is no ideal process and most organizations have developed their software development processes". But one does not want to develop one from scratch, as one then loses the advantages of using a standardized methodology, described in Section 2.1.1. Here is where one can adjust the selected methodology to fit one's context. "Generally, you need to find a balance between plan-driven and agile processes"(Sommerville, 2011, p. 28-29) So, to find such as balance, the balance needed for one's context, one can *tailor* the methodology. As Brooks wrote in his book *The Mythical Man-Month* from 1975: "There is no Silver Bullet".

The case presented in Section 1.4 has tailored the LeSS Framework to their context and organization. Results from the research are presented in Chapter 4. To reason about how successful the tailorings that were made in the case have been, one must first give examples of other method tailorings. Therefore, examples of tailoring will be presented in Section 2.3. First, the framework that is tailored in the selected case will be introduced in section Section 2.2.

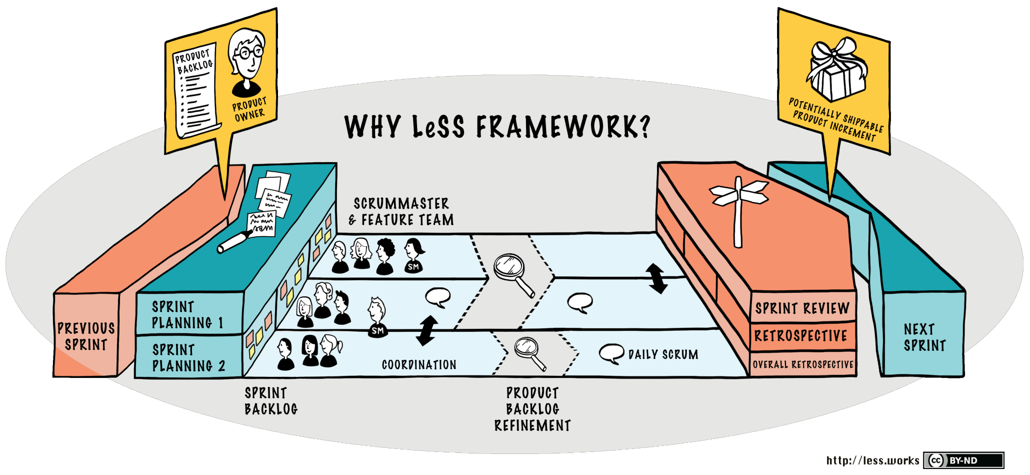

Figure 2.1: Figure by the creators of the LeSS Framework that shows the main elements in the framework. (*LeSS Framework*)

2.2 Large-Scale Scrum (LeSS)

One software development methodology that one can use in multi-team projects is LeSS. It was the inspiration for the methodology used in the case that will be investigated in this case study, introduced in Section 1.4. To be able to reason about the case, understand their activities and how these were tailored from LeSS, there is a need to describe LeSS Framework. Therefore, this section will give an overview of the LeSS Framework, based on the book *Large-Scale Scrum: More with LeSS*. It will mostly use Appendix A of the book, see Larman and Vodde, 2016, p. 333 - 336 to provide an overview of the LeSS Framework, but may also draw from other parts of the book.

LeSS was created by Larman and Vodde, and by the creators called a "framework". LeSS was developed for projects with 2-8 teams, and a variation called LeSS Huge was created for projects with more than eight teams. The overall structure of the LeSS Framework can be seen in Figure 2.1. As it is the regular LeSS Framework that inspired the methodology used in the case, it is the only variation described here.

2.2.1 Structure

LeSS is based around teams. The teams should, like in Scrum, be self-managing, cross-functional, co-located and long-lived. They should also be feature-oriented, and not oriented around components. This necessitates a vertical architectural decomposition of the system to be developed, in contrast to a horizontal one, according to Buchmann, Nord, and Ozakaya. See Figure 2.2.

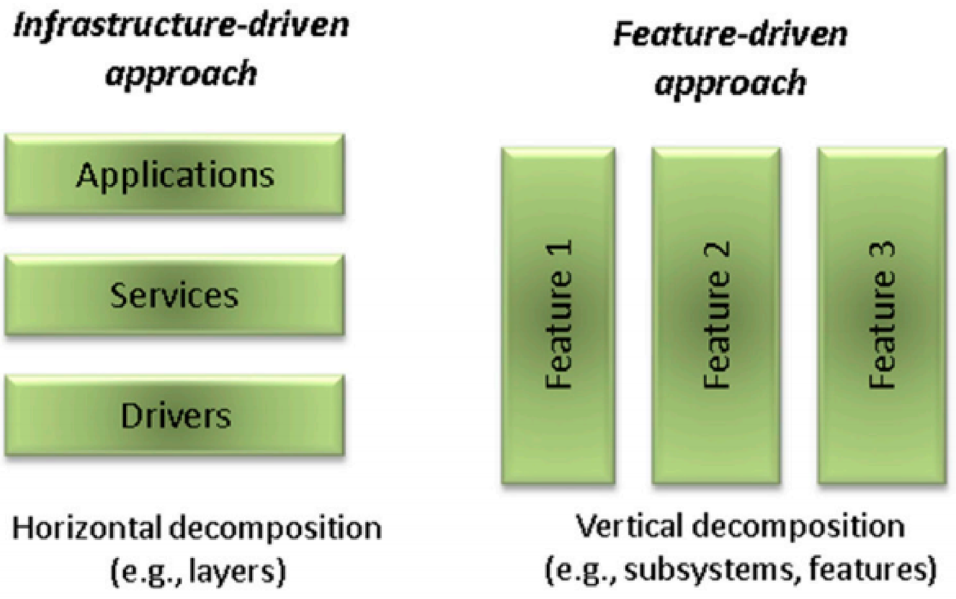
Infrastructure-driven approach
Feature-driven approach
Applications
Services
Drivers
Feature 1
Feature 2
Feature 3
Horizontal decomposition (e.g., layers)
Vertical decomposition (e.g., subsystems, features)

Figure 2.2: Illustration of horizontal and vertical decomposition of a system. (Buchmann, Nord, and Ozakaya, 2012)

LeSS bases itself on one common Product Backlog, called Backlog for short. It is a prioritized list that consists of everything that is to be done. The Product Owner (PO) prioritizes the backlog, which means the items should be understandable to the PO as well as the developers. The prioritization is done according to a *product definition*, which defines what the product is meant to be when it is done. The product definition should involve some end-user or customer.

As the items in the Backlog are understandable to everyone, they may not be practical as items that the developers can work on, i.e. if the items are written as user stories, it may be difficult for a developer to pick a user story and simply implement it. Therefore the teams refine or groom the items into more practical tasks. This process also includes clarification from the PO. This is done in a Product Backlog Refinement (PBR) meeting, described in Section 2.2.3.

LeSS is oriented around time-boxes of 1-4 weeks called Sprints. Before each of these Sprints, the teams conduct a Sprint Planning to define what they are going to do for that Sprint. This results in a Sprint Backlog for each team, which is a prioritized list of items that show what the team has committed to, for that Sprint. I.e. the team picks some items from the complete Backlog and says to the PO that they think they can complete those items in the next Sprint. No one can change that commitment, other than the team itself or the PO. This removes the distraction of stakeholders producing new requirements during development of other requirements. The Sprint Planning is further described in Section 2.2.3.

When team members pick items from the Sprint Backlog, there must be a clear definition of when the item is done. Therefore there is *one* Definition of Done that is common for all teams. The Definition of Done is simply a definition of what it means for an item to be done. This enables a developer to *know* when the item she is working on is done. Teams can expand on the common definition.

The Definition of Done should be defined in such a way that each Sprint can lead to one shippable version of the product, i.e. the Definition of Done should include validation and acceptance tests. The Sprints are common for all teams, i.e. they all start and stop a Sprint at the same time.

The next sections, Section 2.2.2 and Section 2.2.3, will further explore the various roles and activities of LeSS.

2.2.2 Roles

The LeSS Framework has the following roles:

Product Owner (PO): The PO decides what is going to be accomplished, i.e. what is to be developed: Risk and what is possible is taken into account, and the PO uses this to add or remove to the Product Backlog, and prioritize it. The PO has the responsibility for the product as a whole, and the direction it should take. The PO delegates as much clarification work as possible to the teams. This frees up time for the PO, and it allows direct communication between the teams and the stakeholders.

Area Product Owner (APO): An Area Product Owner (APO) assists the PO in taking decisions about the direction of the product, but where the PO has the responsibility for the product as a whole, the APO has the responsibility for one requirement area. One requirement area should have at least four teams because with 1-2 teams the APO role " . . . mutates into an item-clarification role, a kind of analyst or specification writer, rather than someone with a strategic and profit focus towards a major market area." (Larman and Vodde, 2016, p. 194-195)

Team member: The team makes what the PO decides that they should make. The rule of thumb is that the team consist of 3-9 people. The team refines and estimates the Product Backlog so that it is clear what the items there mean to them, and so that the PO knows how much time an item will take. They do not estimate tasks in terms of hours, but relative size.

Scrum Master: The Scrum Master teaches the LeSS Framework to the team, coaches them and removes obstacles or hindrances that slow the team down. The Scrum Master is responsible for a well-working LeSS adoption, and is one of two "meta-feedback loops" for seeing if it the adoption works well. The other meta-feedback loop is the Retrospective, described in Section 2.2.3. One Scrum Master can serve 1-3 teams, but it is a full-time position to be a Scrum Master. (Larman and Vodde, 2016, p. 135-137)

2.2.3 Activities

The LeSS Framework has the following activities:

Sprint Planning: Sprint Planning is a meeting where the team, the PO and the Scrum Master plans the Sprint. The team looks at the top of the Product Backlog and decides how much they can accomplish in the next Sprint. Everyone decides on a Sprint Goal, which is "what everyone wants to accomplish with the Sprint".

In LeSS, the Sprint Planning has two parts. In Sprint Planning One all teams takes part. Then there is the Sprint Planning Two, which is usually done by each team separately.

Sprint Planning One is attended by the PO and the teams or representatives for the teams. The goal is to select the items from the Backlog that each team is going to do for that Sprint. In this meeting, they also want to "identify opportunities to work together".

Sprint Planning Two, in turn, is done by each team, where the design their own Sprint Backlog based on the items they are given in Sprint Planning One.

Daily Scrum: Each team meets for their own Daily Scrum, also known as the "Daily Standup" or just "standup." In the Daily Scrum, each team member should answer three questions (Sutherland, 2014, p. 237):

1. *What did you do yesterday that helped the team finish the Sprint?*
2. *What will you do today to help the team finish the Sprint?*
3. *Is there any obstacle blocking you or the team from achieving the Sprint Goal?*

Product Backlog Refinement (PBR): The PBR is a meeting that is done for each team, where they go over the items in the Backlog which they are likely to do in the future, and refine them. The concept of refining the Backlog is also known as *grooming*, mentioned in Section 2.2.1. For this reason some people call the meeting where one refines the Backlog for a "backlog grooming meeting". To *refine* the items, means to specify them in such a way that the item can be worked on.

In the LeSS Framework it is recommended that one sometimes does PBRs with multiple or all teams, to increase understanding and enable coordination across them.

Sprint Demo/Sprint Review: A meeting where the team(s) demonstrate the shippable iteration of the product that they have developed, and where stakeholders are allowed to give feedback. This allows the team(s) to adjust the product after a Sprint, so that it meets the needs of the stakeholders.

Sprint Retrospective: This meeting is for the team members to see what worked well and what did not work well in the last Sprint. The Retrospective works as a meta-feedback loop for seeing if the LeSS Framework works, along with the Scrum Master. This is because this arena is designed to identify how the team can improve. Challenges are identified in this meeting, and new solutions are suggested. They may also decide to conduct experiments with the process, to see if a suggestion works.

Overall Retrospective: After the Sprint Retrospective that is done for each team, there is an Overall Retrospective meeting where issues across teams and systems can be discussed, and one can come up with possible solutions or experiments to improve. The Scrum Master, PO and representatives from the teams attend.

This section should give the reader a simple overview of the LeSS Framework. It is limited, but it will cover the relevant elements for the results, presented in Chapter 4. As mentioned, this overview is based on *Large-Scale Scrum: More with LeSS* by Larman and Vodde (2016), which can be recommended if one wants to learn more about the framework.

2.3 Cases of Tailored Methodologies in Large-Scale Projects

To be able to reason about the findings and the research on the case presented in Section 1.4, it is important to first look at relevant cases, which can put this case into perspective. This section will present two cases that are thought to be useful when discussing the results of this case study, which will be done in Chapter 4. The two cases are:

1. “Tailoring Agile in the Large: Experience and Reflections from a Large-Scale Agile Software Development Project” by Rolland, Mikkelsen, and Næss (2016)
2. “A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case” by Petersen and Wohlin (2009)

This section will provide an overview of each of them so that the reader can make sense of their usage in the discussion in Chapter 4.

2.3.1 Guidelines for Tailoring Agile

Rolland, Mikkelsen, and Næss (2016) describes a case study of a development project in a Governmental organization. The development project ran for 3.5 years, with 120 participants. Based on the case study, they developed six guidelines for tailoring agile methods. They can be summarized as follows:

1. **Improve inter-team coordination:** Rolland, Mikkelsen, and Næss suggest that the project should experiment with practices that "highlight functional and technical inter-dependencies in the software being developed". The goal of the guideline is to improve coordination across teams and roles. They also suggest establishing long term Community of Practice (CoP) and short term task forces term to improve inter-team coordination. The "task forces" is a practice they observed in their case, and are temporary groups formed across teams that work on specific pressing problems, usually non-functional things such as security or performance issues.

2. **Facilitate novel practices to emerge:** Rolland, Mikkelsen, and Næss suggest that successful tailorings can come from both bottom-up and top-down initiatives, but that managers should be "wary of trying to enforce predefined tailored practices".

3. **"Record, and move on":** Contractual details should not hinder the work. Rolland, Mikkelsen, and Næss suggest building trust in the development organizations, enabling pragmatic decisions and temporary solutions.

4. **Scale the project in an evolutionary manner:** According to the study, it is important to give the customer time to get accustomed to how the work process works. It is also important to give them training. Both of these points should be done before a ramp-up phase, where the size development organization is increased.

5. **Adjust content in sprints:** In addition to being given time to adjust to the work process before the development organization is scaled up, Rolland, Mikkelsen, and Næss suggests that the customer must be given time to "absorb new information and coordinate requirement elicitation with stakeholders in their organization". A suggested practice for doing this is having technical Sprints, where the customer is left alone.

These guidelines are a result from their one case study. They could be improved or added to, given more case studies. If a case study has used some of these guidelines for their tailoring, the success or failure of the case is interesting as well, as it may show how good the guidelines are.

2.3.2 Implementing Agile and Incremental Practices

Petersen and Wohlin, 2009 looked into the advantages and issues of State of the Art agile methods in a large-scale setting, by conducting a case study. The case consisted of four components and 203 people working on them. 33 of these people were interviewed so that at least two people with each role was interviewed, as far as it was possible.

Through the case, eight advantages and twelve issues were identified, which was mapped to advantages and issues described in State of the Art Scrum and Extreme Programming (XP). The advantages are shown in Table 2.1, and issues are shown in Table 2.2.

The main finding was that introducing practices introduces both advantages and issues. One cannot expect benefits alone. They also found that the literature and empirical research agree on both advantages and issues in agile.

If the same issues can be identified in other cases, then one can use that knowledge and move resources to tackle those specific issues. New issues can give a more nuanced look at the findings of Petersen and Wohlin. The same goes for the advantages: Identifying them in more cases, means increased knowledge on how they emerge. It is also worth noting that, as Petersen and Wohlin also states: New advantages means new issues.

2.4 What is Architecture?

Architecture is often challenging in large projects, as shown by a case study by Dingsøyr et al. (2017), which found architecture as a key challenge. Some issues identified by Petersen and Wohlin (2009), introduced in Section 2.3, were also related to architecture. Architecture is also explored in this case study. To be able to discuss the matter of architecture the term has to be defined, and some key concepts must be introduced. First, Bass, Clements, and Kazman, 2012, p. 30 defines *software architecture* in the following way:

> *The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.*

In other words, the architecture is an abstraction of the system. Why would an abstraction like this be important for software development, especially for large-scale development? Bass, Clements, and Kazman (2012, p. 24) provide 13 reasons for why architecture is important in general. Svorstøl (2016) summed these 13 reasons up in the following way:

> *Architecture (an abstraction of the system) ...*
>
> - *... simplifies and helps communication among developers and stakeholders, i.e. when a new developer starts working, a new feature is to be implemented or cost and schedule is to be discussed.*
> - *... enables developers to reason about overall quality attributes of the system.*
> - *... shows or defines the decisions that are hardest to change about the system.*
> - *... can define the structure of an organization, or vice versa.*

There are three main categories of these abstractions: Module structures, Component-and-connector structures and allocation structures. (Bass, Clements, and Kazman, 2012, p. 10-11) Svorstøl (2016) described the three in the following way:

1. **Module structures:** "Structure that partition the system into implementation units. I.e. one part that handles sending e-mails, while another part has the logic of user accounts." A partitioning unit of this category is named a *module* in this thesis, the same name that Bass, Clements, and Kazman, 2012 uses.

2. **Dynamic structures:** “Shows how elements interact during runtime. That is how runtime structures interact with each other, and with infrastructure. For example, there could be a runtime structure with the responsibility of sending e-mails to users, while another running structure handles user accounts. During runtime, the user account structure interacts with the email structure to send the users e-mails. ” A partitioning unit of this category is named a *component* in this thesis, the same name that Bass, Clements, and Kazman, 2012 uses.

3. **Allocation structures:**: Structures that allocate members of some set A to members of some set B, “i.e. mapping a running element to hardware elements, or allocation of work to developers”.

The three categories show that architecture can be used for communication around a system, as well as help with organizing software developers. The third category is what is most relevant for the context of this thesis, and the findings of the case study. Therefore, there is a need to explain this category in more detail.

To be able to do an allocation of a computer system to a set of developers, one must first decompose the system or the work to be done, into some partition unit. Svorstøl (2016) described the two main types of decomposition in the following way:

> *. . . let us look at two ways of doing decomposition: If there is a system X that is to be developed by a team of Y developers, there are two basic decompositions for the system, so that each developer has his or hers own responsibility:*
>
> **Vertical decomposition:** *Each developer has responsibility for every piece of implementation when developing a feature, i.e. everyone is working fullstack. It could also mean that the system is separated into subsystems, that each serves a set of features.*
>
> **Horizontal decomposition:** *Each developer has responsibilities based on system infrastructure. This means that he developers are organized by the architectural needs of the system. A typical example is a system with three layers: The UX, business logic and database logic. With a strict horizontal decomposition, [each developer] would work on only one of*

the three layers, which means a feature could be implemented by more than one developer.

Svorstøl (2016) then notes that these are the extremes of the decomposition. In a complex system, one usually does not have simply one of these types, but a hybrid.

Now that architecture and allocation structures have been defined, some models for handling architecture in large agile projects can be presented in Section 2.5.

2.5 Architecture in Large-Scale Agile

In Section 2.4, the term *software architecture* was defined, its categories introduced, and the two types of decomposition, horizontal and vertical, was explained. This section will explain some ways of organizing architects and architectural work in large projects.

In 2014 Eckstein (2014) released an article that describes a model for organizing architectural work in a large project. The model bases itself on the complexity of the system, and the uncertainty in the project. This relationship gives the stability of the architecture: A project with a high number of changes and a lot of uncertainty has a more unstable architecture. The relationship is shown in Figure 2.3.

In Figure 2.3, one can see three states that the system can be in: (1) Unstable and complex. (2) Adapting. (3) Stable and uncomplex. Eckstein proposes models for organizing architectural work, one for each of the three states, under some conditions. The conditions for the project itself are that there are self-organizing, cross-functional teams, which are oriented around business value. These are called feature teams, and are defined by Larman and Vodde in their book *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*:

A feature team . . . is a long-lived, cross-functional, cross-component team that completes many end-to-end customer features one-by-one. (Larman and Vodde, 2010, p. 549)

The definition falls in line with the pre-condition for teams in the LeSS Framework, defined in Section 2.2.Given a project with feature teams, this model determines if that project can be designed up front or not. See figure 2.3. With this stability in mind, she describes models that can be employed in different project contexts. These models will now be presented, through the words of Svorstøl (2016):

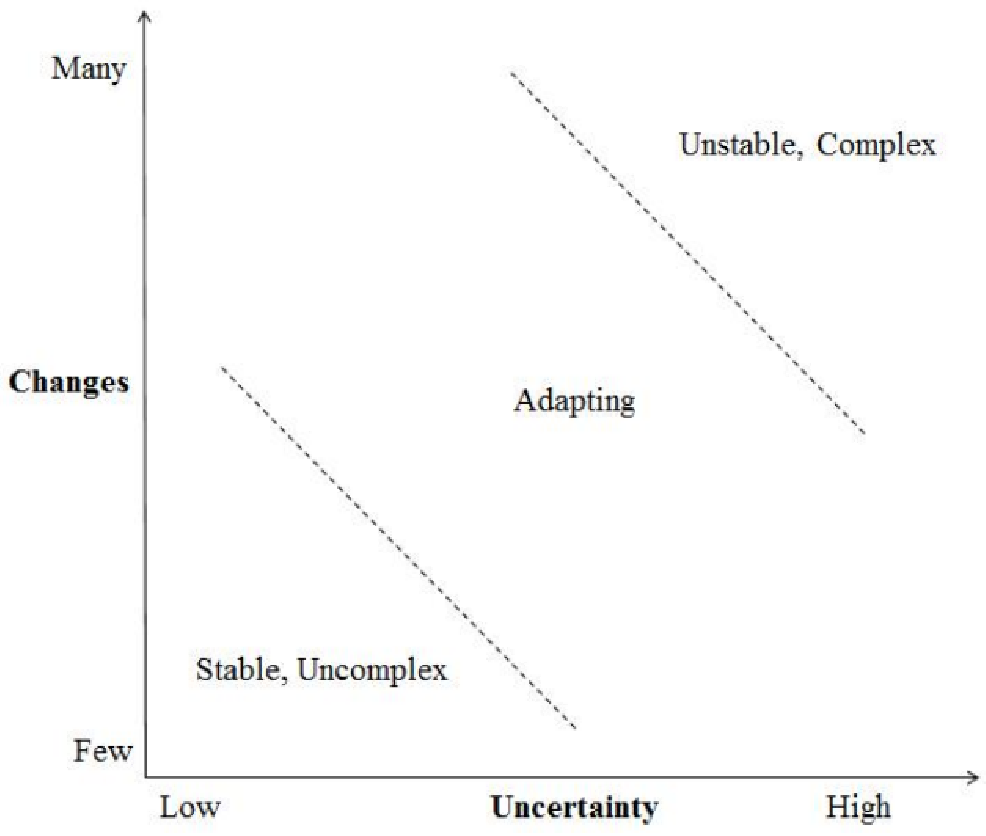

Figure 2.3: The relationship between changes and uncertainty that can be used to determine project complexity, as described by Eckstein. (Eckstein, 2014)

Community of Practice (CoP): This model was originally proposed by Larman and Vodde (2010). In this model, each feature team serves the role of an architect. Not all team members take part in the architectural activities, but those that are interested do. These are a part of the CoP. I.e. the CoP are made up of various team members that want to work with architecture. The CoP can meet to discuss architectural decisions,

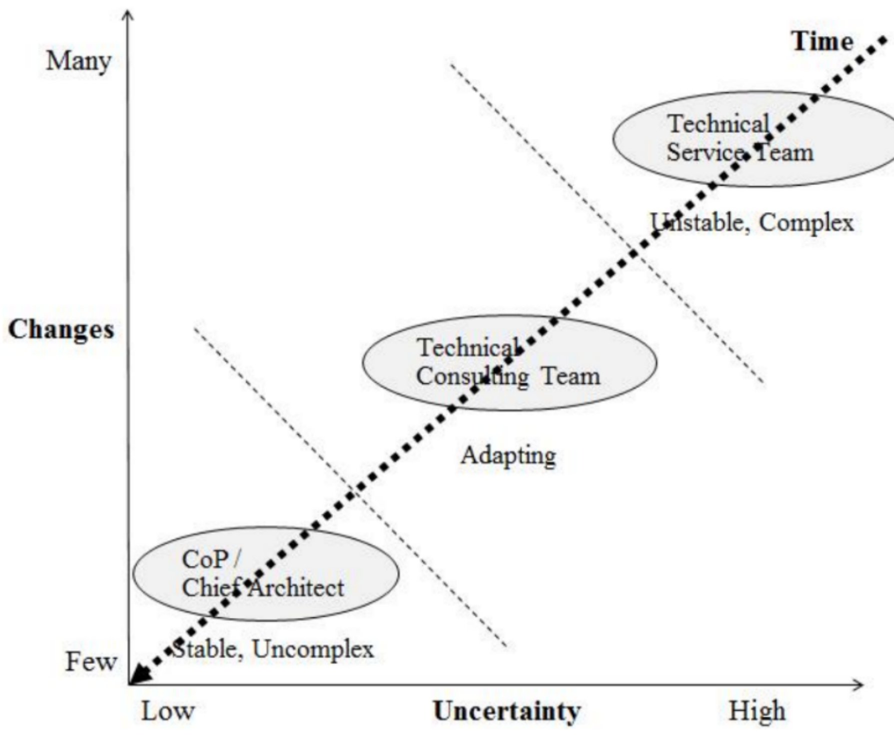

Figure 2.4: Figure that describes how Eckstein sees a project change in complexity and changes over time. (Eckstein, 2014)

and meet regularly to discuss changes or possible improvements. It may seem optimistic that the CoP always will meet agreement around decisions, but it is important to note that as the model is meant

One may argue that this is a utopia of a model, as it suggests that all of the people in this CoP will reach agreement on all architectural decisions. That being said, Eckstein suggests this model for projects which are pretty much straight forward, so there's the assumption that not that many architectural decisions to be made, as the architecture and project are already stable.

Chief architect: For stable and uncomplex projects, Eckstein suggest having a single person as a chief architect, which provides architectural support to the feature teams. The person should have architectural experience, technical knowledge and should be socially skilled. According to the Eckstein the person should do most the architectural work wandering around. Having a chief architect as opposed to a CoP removes the challenge of not reaching agreement on architectural decisions, but the person may also have an enormous pressure and responsibility. If the chief architect must quit, e.g. has to move, it is a huge loss to the project. That means that for larger projects, a CoP may be better, or a combination of the two models.

Technical Service Team (TST): In a project with an unstable architecture, Eckstein suggests using a Technical Service Team (TST), which is a team that together works as the chief architect. The team provides architectural services to the feature teams. The feature teams provides a PO to the TST

Technical Consulting Team (TCT): When supporting an adaptive architecture (medium certainty and changes, the middle of Figure 2.4), Eckstein suggest a Technical Consulting Team (TCT). It is a sort of mix of the previous models: For each agile iteration, each member of the TCT will offer their services to one feature team. Which feature team the TCT-member works with, may vary between iterations based on the needs of all the feature teams. According to Eckstein, the number of members of the TCT should be lower than the total number of the feature teams.

According to Eckstein, a project can move through these models, i.e. change model with changing complexity and the stability of the architecture. Eckstein shows this with figure Figure 2.4. But architectural stability is important in large projects, nevertheless. This is supported by Kruchten (2013) as well, where he argues that the context of the project is what defines the success of the project when one scales agile methods. Part of this context is the stability of the architecture, such as how much is defined up front. This is also supported by Dingsøyr et al. (2016). The point is that even though there are frameworks and models, using them is not a measure of success. As Kruchten (2013) writes, with the agile definition of Jim Highsmith: "... we define *agility* as 'the ability of an organization to react to changes in its environment faster than the rate of these changes'". Having agile artifacts in a project does not in itself make the project agile. This insight, as well as the models and cases presented in this chapter, will be valuable when presenting the results of this case study in Chapter 4. First, in the next chapter, the the method of research will be presented.

Table 2.1: The agile advantages that Petersen and Wohlin (2009) identified in the case. It is a recreation of Table 6 in that paper. In the paper, the advantages CA03-CA08 were also mapped to advantages in state of the art methods.

ID	**Description**
CA01	Small projects and projects allow to implement and release requirements packages fast which leads to reduction of requirements volatility in projects.
CA02	The waste of not used work (requirements documented, components implemented, etc.) is reduced as small packages started are always implemented.
CA03	Requirements in requirements packages are precise and due to the small scope estimates for the package are accurate.
CA04	Small teams with people having different roles only require small amounts of documentation as it is replaced with direct communication facilitating, learning and understanding for each other.
CA05	Frequent integration and deliveries to subsystem test Latest System Version (LSV) allows design to receive early and frequent feedback on their work.
CA06	Rework caused by faults is reduced as testing priorities are made more clear due to prioritized features, and that testers as well as designer closely together
CA07	Time of testers is used more efficiently as in small teams as testing and design can be easily parallelized due to short ways of communication between designers and testers (instant testing).
CA08	Testing in the LSV makes problems and successes transparent (testing and integration per package) and thus generates high incentives for designers to deliver high quality.

Table 2.2: The agile issues that Petersen and Wohlin (2009) identified in the case. It is a recreation of Table 7 in that paper. In the paper, the issues CI06, CI07, CI11 and CI12 were also mapped to issues in state of the art methods.

ID	**Description**
CI01	Handover from requirements to design takes time due to complex decision processes.
CI02	The priority list is essential in the company's model to work and is hard to create and maintain.
CI03	Design has free capacity due to the long lead times as in requirements engineering complex decision making (e.g., due to CI02) takes long time.
CI04	Test coverage reduction within projects due to lack of independent testing and shortage of projects, requiring LSV to compensate coverage.
CI05	The company's process requires to produce too much testing documentation.
CI06	LSV cycle times may extend lead-time for package deliveries as if a package is not ready or rejected by testing it has to wait for the next cycle.
CI07	Making use of the ability of releasing many releases to the market increases maintenance effort as many different versions have to be supported and test environments for different versions have to be recreated.
CI08	Configuration management requires high effort to coordinate the high number of internal releases.
CI09	The development of the configuration environment to select features for customizing solutions takes a long time due to late start of product packaging work and use of sequential programming libraries.
CI10	Product packaging effort is increased as it is still viewed from a technical point of view, but not from a commercial point of view.
CI11	Management overhead due to a high number of teams requiring much coordination and communication between.
CI12	Dependencies rooted in implementation details are hard to identify and not covered in the anatomy plan

Chapter 3

Method for Data Generation and Analysis

The previous chapters have motivated the need for researching method tailoring, presented research questions, relevant case studies and theory. To answer the research questions, and to investigate the method tailoring in the case, there is first a need to describe the research strategy that was used, i.e. describe the descriptive case study. This chapter will base itself on the description of a case study by Oates (2006, Ch. 10) to describe this research strategy. First this chapter will explain the research strategy itself, before explaining its use in this project.

3.1 The Case Study as the Research Strategy

As mentioned, this chapter bases itself on Oates (2006), and he defines the "case study" in the following way:

> *A case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between the phenomenon and context are not clearly evident.*

He further states that the case study is characterized by the following:

- Focus on depth rather than breadth, i.e. the goal is to obtain as much information as possible about a single instance.
- Natural setting, i.e. the case existed before the researchers came, and is not a set-up.
- Holistic study, i.e. it focuses less on individual factors, and more on “the complexity of relationships and processes and how they are interconnected”.
- Number of sources, e.g. it tries to interview multiple people about multiple topics, rather than one person about one topic.

As this is a case study, this will be characteristics of the research. This will be shown throughout this chapter, and summarized in Section 3.7.

3.2 Planning the Case Study

Through his book, Oates provides insight into the various choices one must do when one selects case study as the research strategy. These are:

1. The type of case study
2. Approach to time in a case study
3. Approach to selecting cases
4. Approach to generalizations in the study
5. Selection of data generation methods

This section will explain the choices that were made within these five points, and why those choices were made.

3.2.1 The Type of Case Study

As stated, this is a descriptive case study. A "descriptive case study" is one of three basic types case studies. Oates (2006, p. 145) defines them this way:

Exploratory study: "Used to define the questions or hypotheses to be used in a subsequent study. It is used to help a researcher understand a research problem. "

Descriptive study: "leads to a rich, detailed analysis of a particular phenomenon and its context. The analysis tells a story, including discussion of what occurred and how different people perceive what occurred."

Explanatory study: "Goes further than a descriptive study in trying to explain why events happened as they did or particular outcomes occurred. The case study analysis seeks to identify the multiple, often inter-linked factors that had an effect, or compares what was found in the case to theories from the literature in order to see whether one theory matches the case better than others."

The type "descriptive case study" was chosen because of the following reasons:

- There are not any planned subsequent studies, and there exist other exploratory studies one can use to explore the area. This rules out the exploratory case study.
- There was a time constraint associated with the project, and the necessary extra depth of the explanatory case would take more time, e.g. to be able to research relevant models.

3.2.2 Approach to Time

Oates (2006, p. 144) writes that case studies can have one of three approaches to time:

Historical study: "...examines what happened in the past by asking people what they remember about earlier events and analyzing documents produced at the time."

Short-term or contemporary study: "...examines what is occurring in the case now."

Longitudinal study: "...involves the researcher investigating the case over time, anything from a month to several years, analyzing those processes and relationships that are continuous and those that change."

Due to the time limitations of the project, a longitudinal study was not a possibility as the author also had to have time to both analyze the data, and write about it.

This left a historical or short-term study. The project that became the selected case had not gone on for such a long time, therefore a historical case was not possible. The approach to time that was selected was therefore a short-term/contemporary, which makes this a short-term/contemporary descriptive case study.

3.2.3 Selecting a Case

An important factor in the case study, is selecting the particular case to study. According to Oates (2006, p.144-145), one can base this choice on one of five things:

Typical instance: The case is selected because it is a typical one, and can be representative for other cases.

Extreme instance: The case represents an extreme to other cases, and is therefore selected to provide a "a contrast with the norm".

Test-bed for theory: The case is selected because it has properties that makes it especially suitable for testing a certain theory.

Convenience: People in the case has agreed to give you access to the case, which makes it a convenient choice in terms of time and other resources.

Unique opportunity: The opportunity to study the selected case was an unique opportunity, i.e. the researcher met the right person or was at the right place at the right time.

The case selection in this case study was based on an unique opportunity, combined with convenience. The author worked part time at a consultancy firm, and asked people in the company with if they knew of a project that was currently running with multiple teams. This action itself was done out of convenience, because the colleagues were easily available, and the author knew from previous interactions that many of them were working with customers that ran projects with multiple teams, i.e. large projects. One of the colleague worked on such a project. Among the projects that the authors colleagues worked on, this one stood out because of the following reasons:

- It was said to have at least 3 development teams, with around 20 developers in total
- It was said to have around 50 people involved
- The project was conducted by a large Norwegian, private company in the financial industry, that operated in multiple countries.

The above stated reasons made it seem that the project would be able to answer the research questions, defined in Section 1.3.

The colleague that worked on the project, was able to put the author in contact with the people responsible for the project. The people responsible for the project presented the author with the unique opportunity of full access to that project. This was also quite convenient, as there had already gone some resources into finding the case, and the time available for doing the research was limited. Therefore the author selected the case and made the necessary arrangements.

3.2.4 Approach to Generalizations

According to Oates (2006, p. 145-146), the key to generate "broader conclusions that are relevant beyond the case itself", is generalizations. According to Oates (2006) there are four approaches to generalizations one can have use in a case study: Concept, theory, implications and rich insight. These can also be combined. He explains them in the following way:

Concept: "A concept is a new idea or notion that merges from the analysis, and which sometimes may even require a new word to be added to the vocabulary of the research discipline."

Theory: "A theory is a coherent collection of concepts and propositions with an underlying world-view."

Implications: "Implications arising from a case study are suggestions about what might happen in other similar instances, possible with specific recommendations for actions."

Rich insight: "Rich insight is what we might glean from reading a case study that does not fit neatly into the three categories of concept, theory or implications, but nevertheless gives us important new understanding about a situation."

The approach to generalizations in this case study is to give rich insight into how a large project may be conducted with agile methods. The reason that this approach was selected, rather than one of the other approaches, was that no clear theory or concepts emerged through analysis. With the goal of describing the situation in the case as clear as possible, it may provide new insight into how large projects work in an organization, and how they can be improved. Implications for future research may arise.

3.2.5 Selecting Method for Data Generation

The selected data generation method was semi-structured interviews. The reason for the choice was that it enables the interviewer to touch upon various themes, and ask follow-up questions as needed, as well as come up with entirely new questions which may bring light to the themes in new ways. This, in turn, lets the interviewee speak freely and in detail about the themes that are brought up, and introduce new and related issues. (Oates, 2006, p. 188) It is especially suitable for this case as little was known about the case beforehand. The method would let the project be explored through the interviews, and result in more nuanced data. This would be far more difficult to achieve with e.g. a questionnaire.

The planning and conduction of the interviews themselves is further described in Section 3.3.

3.2.6 Analyzing Textual Data

As the method of data generation was semi-structured interviews, the data to be analyzed was going to be qualitative textual data. Oates (2006, Ch. 18) describes the process of analyzing qualitative textual data, and recommends reading through the data identifying segments that are either:

1. Not related to the research questions or context

2. Explains or describes the research context
3. Appear to be relevant to the research questions

Segments that are not related to the project are dismissed (item 1 above). Those that are descriptive and those that seem relevant to the research questions (item 2 and item 3 above) are marked with a category, so that those segments can be used to describe or explore the theme related to the category, later on.

To be able to do this, one must have an approach to the selection of themes, or categories. The selection of approach is described in Section 3.2.6.1, while the planning and conduction of the data analysis is further described in Section 3.6.

3.2.6.1 Selecting an Approach for Category Selection

Before the data analysis could begin, there was a need for a initial selection of categories. Oates, 2006, p. 269 describes two approaches to this selection: Deductive and inductive. Of these two, the inductive approach was selected. This means that the categories were based on the textual data, not a model found in the literature. The approach was selected because little was known about the project up front. This meant that selecting a model for the deductive approach brought with it the risk of not being able to shed light on interesting themes and findings in the data, as findings would not necessarily fit within the model that was selected.

3.3 Planning Interviews for Data Generation

The previous section explained the overall plan for the research, and the various choices that had to be made before the case study could be conducted. When those choices were made, the research process was done through the following steps:

1. Introductory analysis of the project to select interviewees
2. Create an interview guide for the interviews

3. Schedule the interviews
4. Conduct the interviews
5. Transcribe the interviews
6. Analyze the transcribed interview data by grouping statements into categories/themes

The results of the research is a description of the project and organization, within the identified themes. The results will be presented in Chapter 4.

The following sections will describe the details of the steps above to explain the basis for the results, and so that the research may be reproduced.

3.3.1 Introductory Analysis

Before interviews could be conducted, there was a need to both select interviewees and to create a interview guide. By talking to the contact in the project, a simple overview of the project members and their roles was made available. Because there was limited time, there had to be a limit to the number of interviews that could be conducted. It was decided that the minimum was to interview at least one person of every role in the project. Where there were multiple people with the same role, for instance where teams each had their own person with the role, the minimum was to interview one person with the role in each team. In other words: The minimum was to cover all roles in all teams. This would provide a balance between the time constraint and the need for a rich data material. Other than that, the aim was to conduct as many interviews as possible, within the constraints of the project timeline, as well as the time of the interviewees.

3.3.2 Design of Interview Guide and Selection of Interviewees

Initial talks with the company, before the company agreed to the research project, uncovered that there were to sections in the project organization: The section that did the development and the section that did analyses of the market, and with those analyses produced requirements for the development team. Based on this information there were designed two interview guides: One for those that worked with development, and one for those that worked with analysis and requirement production. The interview guides can be read in appendices Appendix A and Appendix B respectively. The difference in the guides was that the one for those in the development teams, went into methodology in more detail, and also aimed to have the interviewees describe the architecture and documentation in the project. As for the business side, the question's focus was on how they worked with the development teams when it came to the project.

The company made a company computer available to the interviewer so that contact could be made with interviewees, and interviews scheduled through their internal calendar system. The interviewer made contact with 24 people within or related to the project. Out of these, 1 denied an interview giving the reason that as an external consultant, it would not be doable. There were two that the interviewer could not fit in, or find a time that suited them. Out of the 24 people that were contacted, a total of 21 people were therefore interviewed. The selection of interviewees covers all roles in the two main teams, as well as multiple people from the business and analysis section. The relation between the number of people having a role, and how many with that role who were interviewed, is described in Table 3.1, Table 3.2 and Table 3.3. Also there was a team in another country which was assigned maintenance work on one component. The leader of that team was one of the developers in the team, who was interviewed. It was uncovered that there also was a mobile application development team, but none from that team were interviewed.

Table 3.1: Relation between roles, number of people of holding the role and the number of interviews conducted in team 1.

Role	**Number of people with the role**	**Number of interviews conducted**
Developer	7	4
Test lead	1	1
Scrum Master	1	1

3.4 Conducting the Interviews

The interviews were conducted at the office of the project. As the interview guides in Appendix A and Appendix B say, the interview was started with

Table 3.2: Relation between roles, number of people of holding the role and the number of interviews conducted in team 2.

Role	**Number of people with the role**	**Number of interviews conducted**
Developer	6	2
Test lead	1	1
Scrum Master	1	1

Table 3.3: Relation between roles, number of people of holding the role and the number of interviews conducted outside team 1 and 2

Role	**Number of people with the role**	**Number of interviews conducted**
Main project lead	1	1
Architect	2	2
Release Manager	1	1
Tester	1	1
Product Owner	1	1
Area Product Owner	14	5

some background for the project and the interviewer, to give the interviewee confidence in the interviewer. Then the interviewer went through the questions given in the interview guides. The goal of each section was weighted more than the questions themselves, as to be able to ask questions that could uncover more than the questions defined beforehand. This meant the interviewer asked follow-up questions, and/or extra questions, that could give further information about the project. The extra questions, and/or follow-up questions, concerned the same themes as the questions defined in the interview guides.

According to the authors own time tracking, around 70 hours was spent conducting interviews and in the business. The author was allowed to conduct more interviews, and spend more time in the company, but the time constraint of the project meant that this was not possible.

As semi-structured interviews enabled the author to ask extra questions that were not planned, the data generation method in itself showed to give a lot of data, and in that sense enough methods were used. If one had more time, even more data could be generated. This may have nuanced the findings, or given more findings. E.g. one could do surveys in other parts of the organization.

One could also include more data material in the data analysis, described in Section 3.6, such as documentation.

3.5 Transcription of Recorded Interviews

When the interviews were completed, each interview was transcribed, which means the audio recorded was written out as plain text, so that one could read what each person had said. Then the interviews were anonymized and sent to the interviewees so that they could review their statements. The following sections will explain how these steps were conducted.

3.5.1 Tools

For transcribing the interviews, some tooling was used. First, the transcription was done using LibreOffice Writer[1] with VLC Media Player[2] to play the recordings. Using these tools was not an optimal solution, and the transcription went slowly. Later on Wreally Transcribe[3] was discovered, and was used instead. The software provided features that made the transcription work go faster.

3.5.2 Anonymization

The interviews were anonymized to ensure that the interviewees felt protected. This meant that a mapping from actual names of people and companies was made, where each was given an ID. After each interview had been transcribed, names of people and companies are replaced with these ID's. This change is not considered to alter the nature of the data in any meaningful way. The table with this mapping is at the time of writing encrypted on the author's computer.

[1]https://www.libreoffice.org/discover/writer/
[2]http://www.videolan.org/vlc/
[3]http://transcribe.wreally.com

3.5.3 Review for Interviewees

After each interview had been transcribed, it was encrypted and sent to the interviewee for review. In other words: Each interviewee has had the opportunity to read through their interview. They could also change any statement the interview had, or add anything they wanted. None of the interviewees used this opportunity. The email that was sent with the encrypted transcribed interview can be read in Appendix C. Note that the e-mail is written in Norwegian and that the hint of what the password of the encrypted document was, is removed. The audio recordings of the interviews were deleted after the email was sent out.

3.6 Data Analysis

As described in Section 3.2, the inductive approach was selected for defining the categories in the data analysis. The following sections describes how this inductive approach was used, and what tools was used to do the analysis.

3.6.1 Categories

The categories used for the analysis is shown in Figure 3.1. There are multiple categories under the categories "Meeting and Activities" and "Roles", but they have been removed from the figure to make it easier to read. "Meeting and Activities" has one sub-category for each identified meeting or activity, while "Roles" has one sub-category for each identified role, where Table 3.4 gives a description for each of the categories. The text was placed into all relevant categories, which means that a segment of text could be placed in multiple categories, in multiple levels of the hierarchy.

3.6.2 Tools for Analysis

To conduct the data analysis, the software QSR NVivo [4] was used. The software is designed for analyzing qualitative data. (Ltd., 2017) It allows

[4]http://www.qsrinternational.com/product

the user to go trough textual data material such as transcribed interviews, and code sections of the documents, or sections of texts, into what's called "nodes". In the context of this thesis, these nodes are the same as "categories" described by Oates, 2006. This coding feature allows the user to gather text or material that is regarding a theme, as nodes are named after themes. The user can also define relationships between nodes. An example of this is to define a node for each role a person can have, and code all text related to that role, into the node with the role name. Then the user can look up the node, and see everything that has to do with that role.

3.6.3 Summarizing the Categories and Describing the Project

After having segments of text grouped into categories, each category was analyzed again. This second analysis was conducted by reading through the data gathered for each category and noting common feelings/themes mentioned by the interviewees. This in turn, is used in Chapter 4 to describe the project.

3.7 Evaluating the Case Study

As described in Section 3.1, a case study has the following characteristics, according to Oates (2006):

- There should be a focus on depth rather than breadth.
- The study should happen in a natural setting.
- It should be a holistic study.
- There should be many sources, e.g. not a single interview but multiple.

These characteristics can be seen in this research. It is shown in the following ways:

Focus on depth rather than breadth: Nothing other than the case has been investigated, and the goal of the interviews has been to uncover as

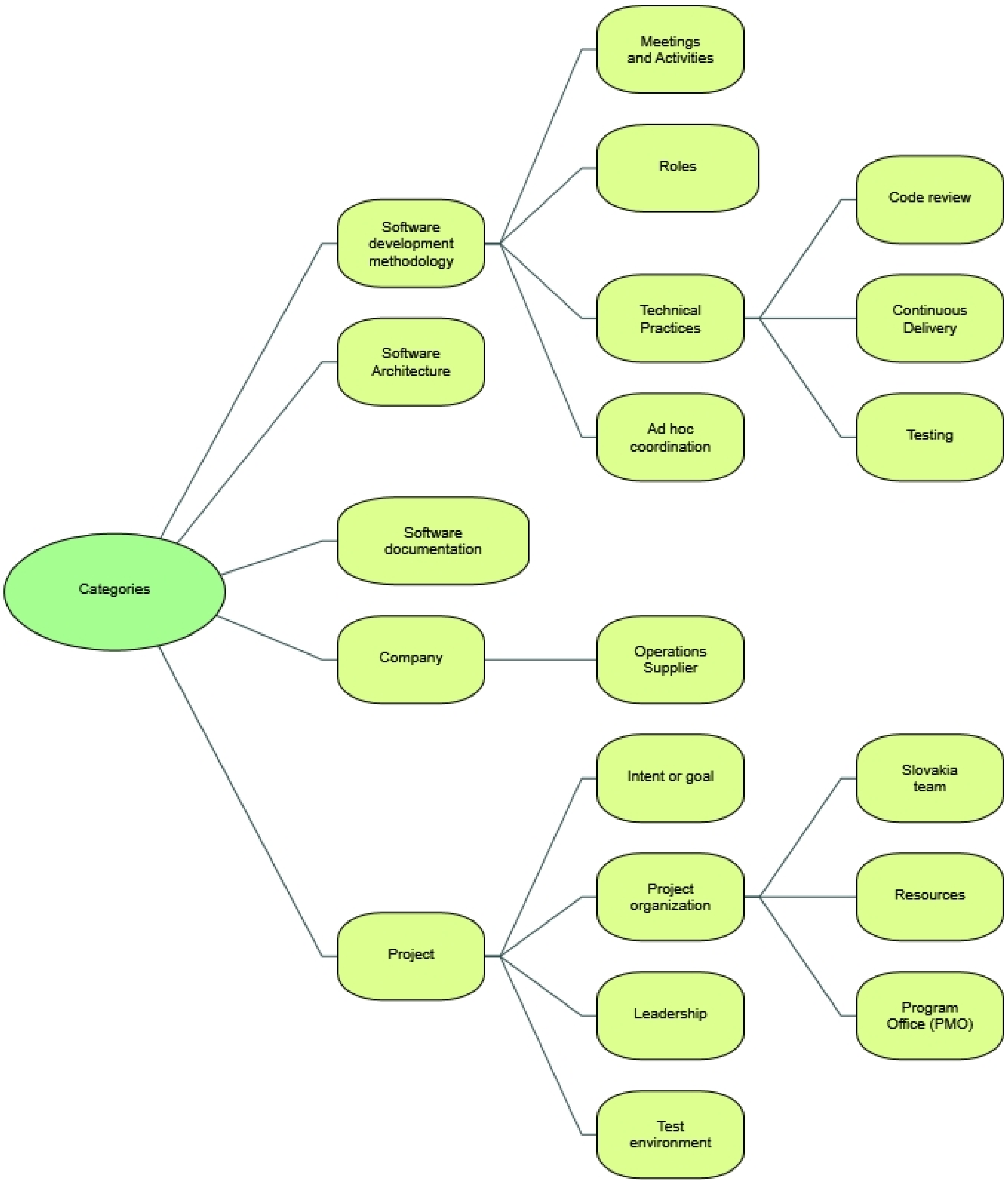

Figure 3.1: Selected categories for the analysis of the interviews, and the hierarchical structure of these categories. Note that the categories under the nodes "Meeting and Activities" and "Roles" have been removed from the figure to make it easier to read.

Table 3.4: Description of categories used in the data analysis of the interviews

Category	**Description**
Software development methodology	Everything regarding the software development methodology used in the project (for definition, see Chapter 2). Sub-categories described in Table 3.5.
Meetings and Activities	The Program Office (PMO) is a organizational unit in the project which is often brought up.
Software architecture	Everything regarding software architecture (see definition in Chapter 2), including that which does not only concerns the project but the company as well.
Software documentation	Everything regarding documentation within the project, as well as the company as a whole. This could be documentation on things, including but limited to, architecture, infrastructure, system setup, requirements and business analysis. This category also includes
Company	For that that is related to company policy, roles, departments, that are not necessarily a part of the project organization.
Operations supplier	The company does not do operations themselves but interacts and works with a supplier which therefore has its own category.
Project	Everything regarding the project that does not have to do with the methodology. Sub-categories described in Table 3.6.

Table 3.5: Description of sub-categories of "Software development methodology", used in the data analysis of the interviews.

Category	**Description**
Meetings and Activities	Category that gathers all meetings and activities in the project as sub-categories.
Roles	Category that gathers all roles in the project as sub-categories.
Technical Practices	Category that gathers relevant technical practices as sub-categories.
Ad hoc coordination	Category for everything that has to do with how they do ad hoc coordination, such as ad hoc meetings, in the project.

Table 3.6: Description of sub-categories of "Project", used in the data analysis of the interviews.

Category	**Description**
Intent or goal	Answers and statements regarding the goal and/or intent of the project.
Project Organization	How the project is organized, i.e. how resources are separated into teams and how they are meant to work with each other.
Slovakia team	Particularities regarding how the project utilizes one team which is located in Slovakia.
Resources	Everything regarding resources such as the number of people working on the project, to have enough competence in certain areas, etc.
Program Office (PMO)	The Program Office (PMO) is a organizational unit in the project which is often brought up.
Leadership	Anything regarding the leadership of the project.
Test environment	The project had its own test environment which was often up for discussion in the interviews.

much as possible about the case. This is shown in both he interviews described Section 3.3, and the analysis described in Section 3.6.

Natural setting: As explained in Section 3.2.3, the case existed before the research began, and has not been altered. It can therefore be described as a natural setting. Section 3.3 explains how semi-structured interviews were used to explore the setting of the case.

Holistic study: As shown in Section 3.3, the questions were open ended to explore relationships and complexities in the case. This can also be seen in the approach to analysis, described in Section 3.6, where the goal was to describe the case within identified areas, and see how they relate. This is can also be seen in the next chapter, where the results are presented.

Number of sources: As described in Section 3.3, numerous people were interviewed. The interview guide touched upon various subjects.

These characteristics can also be seen in the results, which are presented in the next chapter. Based on this, one can evaluate the case study as well conducted.

Chapter 4

Results and Discussion

In Chapter 1 the motivation for this thesis was presented, and in Chapter 2 relevant theory was presented. Then, in Chapter 3, the process for the descriptive case study was described, including data generation and analysis. In this chapter, what was found through the data analysis will be presented, and understood in light of the theory presented in Chapter 2.

First, Section 4.1 will look at how the project is organized in terms of roles, meetings and activities, and discuss this in light of the Large-Scale Scrum (LeSS) Framework that was presented in Section 2.2. In Section 4.2, findings that relate to architecture will be presented and discussed in light of the theory that was presented on architecture in large projects in Section 2.4. Then, Section 4.3 will look at how the findings in this case study relates to those presented in Section 2.3.

After describing and discussing the case, there is a discussion of the significance of the findings in Section 4.5. Finally, Section 4.4 gives short summary of this chapter.

All descriptions of the case that are found in this chapter, are based on the interviews and analysis, described in Chapter 3.

4.1 Project Organization

Through the interviews the following has been found on the organization of the project:

The project is organized into two teams, but there is an additional team located in Bratislava which is doing maintenance work on one of their components. There is also a team that works on a mobile application. This mobile team is not considered a part of the regular development organization in the project, as in operate independently of others in the project. Nevertheless, it is organized under the same project.

The two main teams are receiving tasks that are developed by business analysts, in the business end of the company. There are multiple business analysts, organized around 7 "streams" of work. In the context of this thesis, they will be called areas. Each area has at least one lead, called a "Project Lead." Some have two. In that case, one is from the Department of IT and Development within the company, while the other one is from the business end. Some of these "Project Leads" have business analysts to help them develop requirements, others do not have that. Nevertheless, it is the "Project Leads" that handle all communication with the developers and the developer teams. To be able to look at the project in the context of LeSS, these "Project Leads" will be called Area Product Owners (APOs) in this thesis.

The Project Management Office (PMO) makes the decisions around the project and has members from around the company. In the PMO, there is at least one representative from a group of economists and lawyers, and at least one from a reference group, which are people inside the company which are going to use the systems under development. The PMO is controlled by a "Control group," which has leaders from various business areas as well as departments within the company. The relationship between PMO, the control group, the reference group and law group, as well as to the APOs, is shown in Figure 4.1.

The important finding here is the fact that the project organization relies heavily on the line organization of the company. One example of this that was found in the interviews, is that heads of various departments are in the group that controls the PMO, which is supposed to be in control of the project. Another example that was found is that the APOs are very separate from the development teams. They reported that they may have analyst teams, and sit in another building than the developers, rather than be a part of the

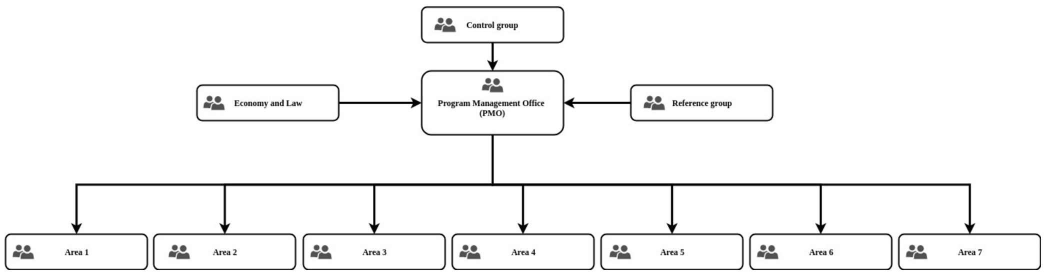

Figure 4.1: Organization of the PMO in the project organization.

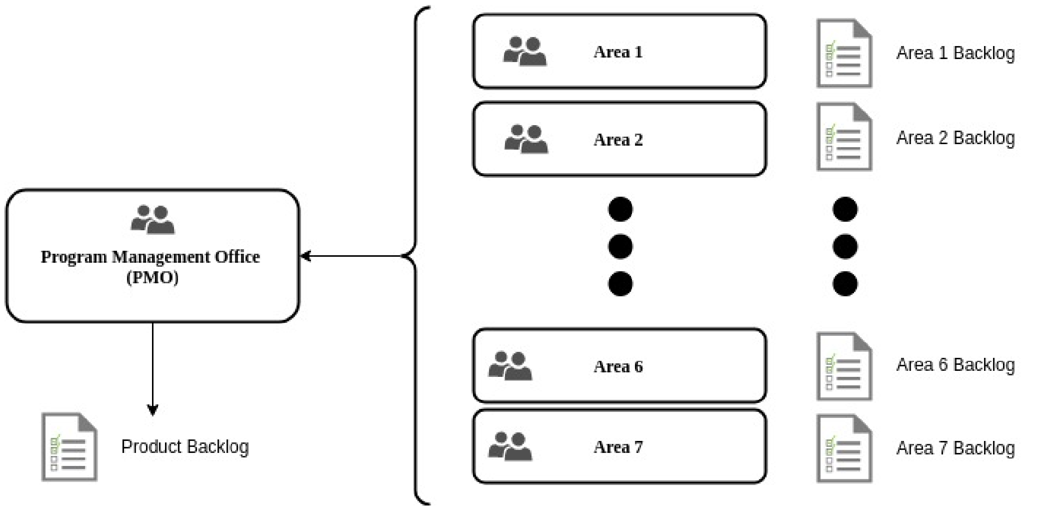

Figure 4.2: Shows how epics flows from areas of the product, lead by APOs, to the PMO. The PMO decides what are the most important epics among those given to them, and based on that creates the the product Backlog.

development teams day to day. The APOs *try* to sit with the developers when they have an epic in for development. It is not always possible for them to sit with the developers, because during development of their epic, they are not relieved of other duties or meetings not relevant to the project.

The findings in the interviews suggest that the project is not organized in the way that LeSS suggests. They have some of the roles, such as Product Owner (PO) and APOs, but the project is not organized as a LeSS project in other ways. The fact of the PMO and control group is evidence of that. These artifacts can be seen as tailorings of LeSS to their enterprise context.

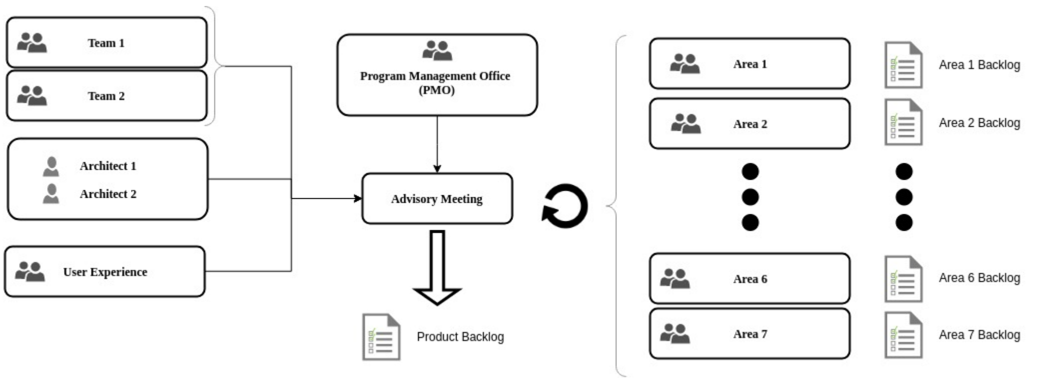

Figure 4.3: Shows how epics are pitched to an advisory meeting, where team representatives, architects and User Experience (UX)-people take part. The APOs may have to work more on epic they pitch, before they can go into development. This back and forth is represented with the wheel between the areas and the advisory meeting.

4.1.1 Roles in the Project

Before presenting findings around areas such as meetings, activities, architecture, and documentation, there is a need to present the roles that the project has so that further results can be understood in the context of those roles. The following overview will also present findings regarding the roles themselves.

Main Project Lead (MPL): The MPL is responsible for the project as a whole. This includes responsibility for ensuring that the teams follow best practices when it comes to development, responsibility for the methodology that the project uses, and to facilitate that usage. The MPL removes obstacles that the team leaders cannot resolve, and pulls in resources when it is needed. There is no defined MPL in LeSS, and therefore this can also be seen as a tailoring.

Team Lead/Scrum Master: There are two team leads, one for each of the main teams. Their title is Scrum Master, but they have more of a project lead role in the sense that they have responsibilities such as approving time sheets and invoices for consultancy firms. As one of them put it “I am a Scrum Master, just with more responsibility for budgets, follow-up work and reporting than what would be natural”. They do not seem to manage the projects but ensure that the team members have what they need. They facilitate most of the meetings that are held within the team and try to remove obstacles for their respective teams. In this thesis, they are referred to as team leads.

Architect: The project has two architects, where one of them is not entirely allocated to the project. The job of these architects is to advise APOs about what can be done regarding epics, and guide developers regarding implementation. They spend a lot of time answering questions regarding interfaces between components and systems. They can be seen as chief architects in the models of Eckstein (2014), where one of them has far more responsibilities than the other. This is further described in Section 4.1.2 and Section 4.1.3.

Release Manager (RM): The Release Manager (RM) is the manager for all releases within the company. This means that the RM has the responsibility of coordinating releases with and between developers, as well as with the operations supplier. This is not a part of LeSS, and the need for the RM comes from how operations is done within the company.

Test Lead: Each team has a test lead who is responsible for ensuring that the systems are properly tested before going into production. The test lead takes part in most meetings with APOs, except the advisory meeting, and asks questions and clarifies so that the acceptance criteria for epics and user stories are unambiguous. Then the test lead plans tests that will check if the acceptance criteria are met.

Tester: The project has one tester, which conducts tests planned by the test leads. The teams have one tester that they share. There was an additional tester for a period but was later scaled back.

Developer: The developers works on developing epics in the Backlog. Each developer works on a specific component. One developer in each team has "Technical Responsibility," which means that the other developers on the team contact him or her regarding technical difficulties with development environments, tools, et cetera.

Product Owner (PO): The PO makes decisions regarding what is the most important epics. The person that used the title PO in this project, is not, in reality, the PO, but rather the "right hand" of a line manager in the company. The line manager has the real responsibility of being the PO. That being said, it seems that in practice this "right hand" is the PO, therefore this person is what is referred to as the PO in this thesis.

Area Product Owner (APO): Each APO is responsible for one "stream" or area of work. They are in reality known as project leads, where the "project" to them is the project of collecting requirements. They usually have analysts that work with them or use people in the organization that is going to use the systems when they are completed.

Most "streams" have two APOs, where one represents the department "IT and Development," while the other represents the business side of the company. Within the business side of the company, they all belong to their section, where they report to their line manager. This section is relevant to what their "stream" is about.

Compared to LeSS, the "streams" can be viewed as requirement areas. LeSS recommends that one requirement area should have at least four teams. The reason for this was that the role " . . . mutates into an item-clarification role, a kind of analyst or specification writer, rather than someone with a strategic and profit focus towards a major market area." (Larman and Vodde, 2016, p. 194-195) Based on the interviews, in may seem that their focus is on specification and analysis, but that the analysis is done to see if a requirement can lead to profit growth. In that sense, Larman and Vodde are both right and wrong when comparing with this case.

Even though the project has most of the roles in LeSS, it seems that the project misses some key concepts. The developers cannot work on all components. The Scrum Masters are not just coaching the teams in agile methods and facilitating meetings, but are spending time on management tasks.

4.1.2 Meetings and Activities

The project has several defined meetings. When it comes to activities, the most common one is software development. This activity was mostly not discussed in the interviews, which were oriented more around project organization. The following meetings have been identified in the project:

Standup: Each team has daily standups. There is no common standup. Every developer attends this standup. APOs also tries to take part in them, when they have one of their epics in development, but they do not always have the opportunity, due to the number of meetings and activities associated with their department. Only developers, architects, and team leaders are allowed to talk during the standup, but APOs can answer questions, or talk with them after the standup. Nevertheless, the APOs feel that these standups are quite useful, even more useful than status meeting that they have with their department. One of them put it this way: "As long as an epic is in development, it makes complete sense to attend. Actually, they are more relevant than these status meetings, because it is sort of there the value, or what we care about, is made."

Developers report that the intent of the standup is to see hindrances early. Everyone reports that it is useful to them to some extent. Some of the developers said that they were annoyed that they sometimes had to spend an excessive amount of time on the daily standup, because other developers took discussions within the standup, instead of taking the time for the arguments after the standup, so that only those involved must spend time on the issue.

The Daily Scrum, also known as the standup, is a part of the LeSS Framework, and is therefore not a tailoring. It seems that they use the questions that are associated with the standup, described in Chapter 2. It also seems that it works as intended, but that there sometimes is necessary with stronger guidelines for what is to be discussed in the standup, so that discussions are taken after it is done.

Sprint or Backlog grooming: When an epic has been approved by the PMO for development, and a team is assigned, it is brought into Backlog grooming. The Backlog grooming is done once or twice each week and lasts 1-2 hours. The goal of the meeting is turning the epic and its user stories into tasks that developers can work on, defining what is to be done and the acceptance criteria for completion. Each team has a Backlog grooming.APOs attends the Backlog grooming meeting when they have an epic ready for development, or in development, i.e. when they have something that needs grooming. Other than that, the developers and the Scrum Master attends. The architect(s) may also attend.

This is the same activity as the Product Backlog Refinement (PBR) in the LeSS Framework, except that there is no common grooming/refinement session: Each team refine their assigned user stories or epics, together with relevant APOs.

In the case, it was reported that these meetings often got very long, even though both team leads aim to invite only the relevant people for each item on the meeting agenda. E.g. the APO is invited to the part of the grooming in which the epic they work with is discussed. The same goes for developers: They are involved when the meeting participants are grooming tasks for components or modules that the developers work on. The reason for this is to protect the time of the developers. One developer reported that there *should* be an architect in every Backlog grooming, to have an overview of the architecture. Test lead also attends the Backlog grooming, to ensure that the acceptance criteria are made and that they are clearly defined.

As for how well the Backlog grooming goes, it is reported by some developers that there are discussions that should have been done earlier on in the process, such as in advisory meetings. They indicate that the effectiveness could be increased. A team lead complained that the APOs was not always engaged, i.e. they were reading emails or were otherwise mentally not present. According to the team lead, this made the meetings take more time than necessary. One developer thought it was strange that they set a date for delivery for an epic in this meeting, but that nothing was estimated. This is not in line with the estimation policiy in LeSS.

Some developers report that this meeting not always is used to groom the Backlog, but rather that they are an arena where developers report the status of the task they are work on and ask additional questions to the APOs. It is also reported that sometimes the APOs reprioritize in these meetings. One of the developers on team 1 said:

> *We see how things are going, and then business comes into the picture, and it often happens that things change. They have thought about it, and found new requirements they want us to include. So it is sort of a reprioritization of things: Either we remove or add stuff, and then we have to move release dates.*

One of the test leads reported that there may show up additional user stories in the middle of the Sprint which may mean that something is missing in their the process of grooming. Combined with the reports from the above mentioned developer, that the grooming meeting is more of a status meeting, it may seem that there is no real grooming meeting. This does not seem like an active choice, i.e. a tailoring of LeSS, but more of a negative effect of the implementation. It may also be that the implementation simply is unsuccessful.

Sprint planning: Each of the two main teams has their Sprint planning. It is unknown if the Bratislava or mobile team has their Sprint planning. According to the team leaders, the purpose of the Sprint planning is to plan for the next Sprint, but the developers perceive it more as a status meeting, where they go through epics and report how they are doing on them. When asked what the purpose of the Sprint planning was, one developer answered: “To see what is left to do. To see what I should be working on for the next weeks”, while another answered “In this project it has not been that structured. We go through all the tasks and report on their status, and that’s that.” When asked about the purpose of the Sprint planning, one developer said: “The purpose is to plan the next Sprint, and I don’t feel that I have any other role than developer there. The tasks should’ve been groomed and ready, but that is not always the case.”

The team leaders are not entirely happy with the planning, as they also experience it as more status reporting. At the time of the interviews, they reported a wish to move away from LeSS to Kanban, due to the fact they did not use Sprints as defined in neither LeSS or Scrum. The MPL on the other hand, said the following about the Sprint planning: “. . . it’s where what is to be solved is presented by the business people, or the Scrum Master, and the team tries to find out how that can be solved.” This indicates that the meeting is not working as intended.

The responses regarding the Sprint planning and Sprint grooming makes them both seem like status meetings, which may indicate that there is only grooming, and that interviewees are mixing the two up. The answers may also indicate that there is no real Sprint. Nevertheless it is difficult to see evidence for them using LeSS here.

Sprint demonstration/reivew: Every other week the teams have a Sprint demonstration, also known as Sprint review, where the team demonstrates what they have implemented in the last Sprint, and attendees can give feedback regarding how it works. All stakeholders are invited, such as the PO, APOs, team leaders, people working with UX, developers and others within the company, such as leaders and managers who want to look at the system. The PO said that it was a “top heavy meeting invite.” The demo itself takes around 20 minutes. Each team has their demo. The demos seem to have the same structure for both teams.

Based on the interviews, there is not much to say about this meeting. It appears to serve it’s purpose, and no one seems to think it takes too little or too much time.

The fact that there are two meetings, one for each team, marks a striking difference to LeSS Framework. Other than that, the meeting seems to serve its purpose.

Sprint retrospective: Every other week, after the Sprint demo, there is a Sprint retrospective, one for each team. According to one test lead, they are not always done. All the regular team members take part in the retrospective, and sometimes those working with UX also participates. There is no common, i.e. Overall Retrospective with both teams, as described in LeSS.

It seems that the two teams run the retrospectives a little differently. Team 1 reports that the goal is to find challenges or hindrances. One developer from team 1 said this when asked what the purpose of the meeting was:

> *it's to uncover the biggest hindrances. And we've also done it in such a way that one can suggest, for example, changes, or at least say what problems one is experiencing. And then we sort of vote over the largest hindrances, so that the Scrum Master has those until next time.*

So everyone can suggest challenges or hindrances, and then they vote over which ones they should focus on resolving until the next Sprint. The team members find this useful but are frustrated when challenges or hindrances are outside their control, such as an unstable test environment. As a team, they can only report the problem to those responsible within the company. The teams are not completely self-organized and not in complete control of their product. This is not in line with the recommendations from the LeSS Framework.

Team 2 seems to have a more structured approach to the retrospective. All attendees write positive things and possible improvements on Post-It notes, then they categorize all their notes together. The categories are: External, internal or personal. "External" are things happening outside the team, and "internal" are things that are about the team. "Personal" are things that are about a particular person, not about e.g. the team or things inside the company. When everyone has brought up their notes, they group them so that similar notes are together. These groupings create themes, which is then voted over. Themes with the most votes are focus areas for the next sprint. When asked if he found the retrospective useful, one developer from team 2 said: “it is not extremely useful for my work in the short term, but it improves the process around it”. This indicates that the developers understand the importance of the retrospective as a meta-feedback loop, even though they do not feel that it is that useful in their day-to-day work.

Looking at both teams, it seems that attendees feel that challenges and hindrances are followed up on, although one test lead did not feel that this was the case. According to that test lead, the retrospectives were either not carried out, or the results were not followed up on. When asked about the purpose, this test lead answered: “The purpose of the meeting is to uncover areas that can be improved, so that we make our process better. Sort of like an iterative optimization or improvement of the process.” And, if the areas they uncovered were not followed up on, “it is completely useless”. Developers seemed to think that things were followed up on, and that the Retrospective worked well.

The LeSS Framework does not provide rules for how to do the Retrospective, other than that there should be one for each team and one common one. When it comes to implementing LeSS, there is nothing wrong with how the Retrospectives themselves are done. That being said, the fact that the developers cannot deal with the challenges they face themselves, even challenges that are completely technical, is another sign of the teams not being self-managed. Self-managing, cross-functional teams is a premise for successful use of LeSS, according to Larman and Vodde, 2016.

Advisory meeting: The advisory meetings is not described in LeSS, nor Scrum or Kanban. It is unique for this project. As stated in Section 4.1.3, these meetings lets APOs pitch epics to the PMO as a whole, or some representative from the PMO. The representative is the PO. After a pitch, the architects can say something about what is needed to go through with the proposal. The epics may be denied, and the APOs together with the business analysts has to do further analysis before it can go into development. If everything seems ready, the architects may say something about the amount of work that is needed to complete the proposed epic.

The reason for this meeting existing seems to be that earlier the architects experienced that employees of the business side of the company started development project that went across the established architectural choices. As one of the architects stated, “. . . eventually you hit our core systems”. Therefore it was important that architects were consulted before any development endeavor was begun, so that time was not wasted, and existing systems were used where possible. The architects are especially happy with this meeting, as it gives them the power to decide over what is going into development and what does not.

One of the architects stated that, not only did they give feedback on what was possible and what was not, but they even took part in the prioritization of the epics. The architect said:

> *The purpose is to get an overview of the task, to know more about it, try to get an idea about how large the task is. If it is a really big task that gives relatively little benefit, it will be prioritized down. If it is a small task that gives a benefit, it will prioritized up. Then we'll have to decide how far up on the list it goes.*

The same architect stated that they usually had time for around two epics each meeting.

The APOs did not have negative things to say about the meeting and enjoyed clarification about what was needed to be able to develop their proposal.

As both APOs, PO and architects seem happy with this meeting, it indicates that this is a successful meeting, and a successful tailoring. That being said, some developers reported that they feel tasks were not clarified enough when they came into development, which means that the advisory meeting may not work as some developers expect them to work. One of the architects stated that the developers would be included in the meeting if the task was clarified enough beforehand, but none of the developers interviewed seemed to be involved in the meeting. This again shows that the meeting may not work as everyone expects.

Task allocation meeting: The task allocation meeting is a weekly meeting between the MPL and the team leaders to assign epics to teams. According to the MPL is the meeting inspired by LeSS, which has Multi-Team Sprint Planning called Sprint Planning One, described in Section 2.2.3. In LeSS these meetings are attended by multiple representatives from each team, to plan which teams are doing what in the next sprint. In this project, however, they are attended by the leads from each team only. They are also attended by the PO which makes any necessary decisions regarding prioritization. Architects also attend. The attendance is in accordance with those that attend the Sprint Planning One in LeSS, but the team representatives are missing. The meeting is not an arena for the teams to coordinate any cooperation, though. They simply allocate the epics.

The project leads stated that they also report on how far they have come on the epics that are in development, and review if they should move more epics into development. This is also a status report meeting.

There were not reported any positive nor negative feelings regarding the meeting, which can indicate that it simply is necessary. One of the team leads stated that he would rather have it every other week instead of every week.

APO status meeting: This meeting is also known as the "business status meeting," and is held weekly, organized by the administrative leader of the PMO. Every APOs, as well as the team leaders, are attending, addition to those working with UX. Here the team leaders report their progress from the last week. The APOs do what is called a Seven Keys report, which they also have delivered in written form beforehand. The same thing is done for the team leads: They give the same report in the meeting, as they do in written form beforehand. One of the team leads stated that “putting it mildly: it’s a waste of time”. This is the same experience the interviewees have with the meeting. One of the members of the business end of the company said that “I perceive it as extremely boring”.

So despite everyone feeling the meeting is a waste of time, it is still conducted. The reason for this meeting seems to be that the PMO has to report to the control group what was been done in the last week. The control group is, as described in Section 4.1, the leaders of the project in the traditional organization. In other words; this meeting is a product of the traditional structure of the company, where every subordinate has to report to their managers. Instead of the managers reading the reports, the subordinates are gathered to report what they already have written. The number of people attending the meeting seems to make it take more resources than necessary.

This meeting is a tailoring so that the agile methods could fit into the traditional organization, and does not come from the LeSS Framework. The negative reports from the involved, indicate that it is unsuccessful, at least in respect to those doing the reporting. As described in Chapter 3, those that are in the PMO or the control group has not been interviewed, so one cannot know for certain why the meeting is arranged from the data gathered.

APO standup: One of the APOs reported that they APOs had their standup every week. There are two APOs for most of the streams, where each one is from a separate department. It seems that only those from one of these departments have these standups. It is attended by the APOs from that department, and the goal is to coordinate their work and help each other in their work. Note that only one of the APOs reported that they attended this meeting and that the department association is not coupled with their answers. The one APOs reporting attendance at this meeting, enjoyed it.

There is no recommendation for this sort of meeting in LeSS, although the meeting may be viewed as sort of a Community of Practice (CoP) for the APOs. Nevertheless, it may be that this is a successful tailoring, but it is difficult to say as there is only one of the interviewees that reported attendance. This meeting should be further explored, given more resources.

Go/nogo-meeting: The code is put into production at regular intervals, by agreements with the company's operations supplier. When an epic is close to being ready for production, and a production window is approaching, a "Go/nogo-meeting" is called for. The goal of the meeting is to decide if the code produced meets the requirements of the epic, and if it should go into production or not. The relevant developers, the test lead, team lead, relevant APOs, line managers and the release manager in the company attends the meeting. At the meeting, all the attendees say if they think it should go live or not. That means that e.g. the test lead can say that it should not go live, because it is not completely tested, or a developer can say that he or she is not entirely sure that what is needed has been implemented, and that more clarification is necessary. The fact that everyone attending can say if should go live or not, also means that line managers within the company, can say that it *must* go into production. It is the attending APOs that make the final decision, but the line manager is the boss of the APOs.

The go/nogo-meeting is not a part of the LeSS Framework, and is another tailoring. It seems to work as it is intended: For quality assurance, and ensure that they do not have to roll back anything. The reason for the need of the latter assurance, seems to be that they cannot go into production at any time, so one cannot simply put a fix into production, or roll back a release. As mentioned, there are production windows. According to the RM, there were only a weekly opportunity for release one year ahead of the interviews. At the time of the interviews, there were two release days each week. In addition they could "order" extra releases from their operations supplier. The RM is not yet happy about the arrangement, stating that:

> *In large organizations everyone talks about continuous delivery and agile methodologies, but I do not think that everyone understands what it actually means.*

In other words: The RM in the organization feels that there is a long way to go before they are at continuous delivery, and that this meeting is a step in that direction.

Production planning meeting: When it is decided in the Go/nogo-meeting that the company is going live with something, RM holds a production planning meeting to "reduce risk" when the production comes. The risk reduction is done by uncovering dependencies between what is to be put into production. Only developers that have something that is going into production, and the RM, attend this meeting. This makes it a simple coordination meeting.

This meeting is not a part of the LeSS Framework, and is also a tailoring. There are no strong feelings about it, so it may indicate that it serves its purpose, as no one feels its not needed, and no one loves it. It is simply something that must be done, it seems.

The meeting may also be viewed as a ad hoc meeting due to it simplicity.

There were additional meetings identified, but there were not enough data to say anything conclusive about these meetings. For instance, it seems that the APOs has more meetings where they report status, but also meetings with the PMO where they discuss hindrances. There are also some that mentioned coordination meetings, but it is unclear if this is a regular meeting. There is also a lot of ad hoc meetings and coordination, especially between APOs and developers, and developers and architects. This is further discussed in Section 4.1.4 and Section 4.2 respectively.

4.1.3 Task Flow

The APOs develop requirements in the form of epics, and create a list of things they wants developed. These items may not reach the developers at that time, and the APOs has their own lists of these items. The PMO decides what is the most important epics or requirements developed by the APOs, and creates a Product Backlog of epics. This is shown in Figure 4.2. The Product Backlog is reported to have around 10 items.

When an APO is of the opinion that a epic is ready for development, it is brought to an "Advisory meeting" where PMO and architects can give feedback on the epic. Architects reported that this meeting greatly improved the way they worked with APOs as it allowed them to clear up misunderstandings regarding the architecture, and even tell the APOs that the epic needed further analysis. The meeting is described further in Section 4.1.2, and shown in Figure 4.3.

When an epic finally is moved to the main Backlog, the epic is considered by one of the teams. In what is called the "Task Allocation Meeting", the team leads together with the leader for the whole project, called the Main Project Lead (MPL), decides which team is going to develop the epic. In the Backlog grooming the team turns the epic into user stories and tasks. Finally, the tasks are developed, while APOs are available in the area where the development teams sit.

When the task is completed, is marked as ready and brought to a Go/nogo-meeting in which line managers, developers, team lead and test lead evaluate if it is ready and can be put into production. If it gets a "go ahead" from the meeting, it is taken to There has not been identified any Definition of Done.

All of these meetings are described further in Section 4.1.2.

4.1.4 Cooperation between Software and Business Developers

Section 4.1.3 described how epics are developed by APOs and their business analysts then handed over to developers for development. Section 4.1.2 described several meetings where the epics is refined and split up into tasks. When the PMO has moved a epic to the top of the product Backlog, and one of the teams has started the development, there may be a need for further clarification from the APO.

This type of coordination is typically done by the APOs that have something in development, sit in the same area as the developers, and are available for ad hoc meetings, questions, and coordination, regarding what they want to have developed. This is experienced as a useful tool for everyone involved. The challenge is that the APOs does not necessarily sit in that area. Either because they do not prioritize it, or because they have other obligations. A part of this is that the APOs usually sit in another building than the developers. The company has them often placed in the building that holds the business oriented departments, rather than in the building with the IT oriented departments.

Interviewees have also questioned the number of business developers compared to the developers that are developing the IT systems. Both architects and developers feel that there is a huge pressure on functionality being developed, while there is put far more resources into developing the requirements. When asked what was challenging in the project, one of the architects simply said "We are too few.", going on to explain that the resources on the business end should have been used on development.

At the start of the project, there was far more resources, but it was scaled down. Some resources are split between the project of study, and other projects in the company. This is part due to the fact that they are not available all the time. This is because the components they have competence on does not always need changing to complete an epic, and also because of resources. E.g. there are two architects, but only one of them are working full time with the project. The architect working on the project full time seems to has a lot of work to do. All developers say they are going to that architect when they have questions regarding any relation between components.

The same issue has been reported for testing: There is one tester, and two test leads. There was an extra tester at a time, but this tester was removed because there was not enough to do at that time. This was frustrating to the tester that was left, as that tester feels that there is more than enough to do at other times.

The developers also report that they want to take part in solving problems, not to only implement solutions which are presented in epics or user stories. One possible way to enable that, is to place analysts within the teams.

The reason for the varying number of resources is difficult to spot. Nevertheless, it is clear that there is varying or limited understanding of the work process that the developers use, in other parts of the company. Limited understanding may explain some management choices that have been done for the project, such as how it was scaled up and later down.

4.2 Software Architecture

From the interviews it was found that the project does not have its own architecture, as everything is related to what is done in other projects and departments within the company. All the company systems are tied together. As mentioned in Section 4.1.1, the developers are working on separate components. From the interviews it seems that the reason for this, is that the components are so big and complex, that to be able to work efficiently on them, one must work solely on one. That being said, developers stated that they want more documentation and access to other components, so that they an attempt to understand them.

This section will present the results regarding the architecture, and discuss them in light of the theory presented in Chapter 2. First, the overall architecture is presented. Then, in Section 4.2.1, the findings regarding how the teams and architects communicate around architecture is presented, and discussed. Then, how architectural decisions are handled is dealt with in Section 4.2.2. Finally, the fact that the developers feels documentation is important, shows the importance of documentation when it comes to architecture. Therefore, results regarding the documentation is presented in Section 4.2.3.

In the interviews, the developers and architects said that the architecture of the company's systems could be viewed as three layers:

1. **The presentation layer:** This layer consists of web mobile applications. There are several different technologies here.
2. **The service layer:** A mid-layer which relays requests from the presentation layer to the core system, see architectural layer 3. This layer is developed with a Service-Oriented Architecture. It has some business logic, but it is avoided if it is possible. The reason for the development of this layer, was that they wanted to reduce the number of requests directly to layer 3, because of its age and its response time.
3. **The core system/the data layer:** The core system gathers all business logic and data. It is the basis for all functionally that the company's software systems deliver. The reason for this is that the system is around

30 years old, and has been developed over a long period of time, to support all business needs.

The architecture is stable, and this general overview has not been subject to change for some time. There are new components added to each of the layers, and there are changes within modules and components, but the complete layered architecture is stable.

That being said, ti seems that it is complex, in the sense that there are a lot of components and modules, and a lot to learn to be able to understand how it all fits together. Especially the bottom layer, the core system, has been around for around 30 years, and has vast amounts of business logic. This makes being an architect challenging. According to the model by Eckstein, 2014, a development project with a stable and uncomplex architecture could handle architecture through a CoP or a chief architect. The model does not mention what to do in the case you have a complex but stable architecture, an architecture which might not change, but is difficult to understand.

4.2.1 Communicating the Architecture to the Teams

Given the overview of the layers in the previous section, how that architecture is communicated to developers can be explored. The architecture must be communicated to developers so that they can have a meaningful discussion about things such as what is happening in the system and what to implement. The following means for communicating architecture to the developers have been identified:

1. **Ad hoc meetings:** The most important means for communicating about the architecture, is meetings held by need in the office area. One of the architects sits in the same area as the developers, which means the developers can ask questions when needed. The architect will then answer, or take the time to set up a meeting for further exploration. This is reported to be the primary method of communicating architecture to the developers, and in that sense, the architect works as a chief architect that wanders, as in the model by Eckstein, 2014. A huge challenge with this method is that there is only one architect available. As mentioned earlier: The project really has two architects, but only one works on it full time. The other architect mostly takes part in regular meetings,

such as the advisory meeting. The architect that is on the project full time is not always available to the developers as because of meetings such as the advisory meeting. This is especially challenging due to the complexity of the architecture, within the layers. In short: Resources are a bigger problem than this model in itself.

2. **Diagrams on the wiki:** The architect produces sketches and diagrams of the systems, and place them on the wiki. This is something that the whole company has use of, but the resources come from this project in the form of the first architect. The quality of the documentation will be explored further in Section 4.2.3, but limited resources seem to be a limiting factor for this method of communicating the architecture.

3. **Onboarding/introduction when a new developer begins:** The first architect tries to fit in a meeting with new developers to go through the architecture, as a part of the onboarding process. This not seems to be an integrated part of the onboarding process yet, but it is something that they have considered. At this point, it seems that such a walk-through is done by need.

It is clear that these three ways of communicating are limited by the architectural resources in the project: There is only one architectural resource allocated 100% to the project. How the architecture is communicated in itself, generally seems to work well.

4.2.2 Architectural Decisions

Developers and architects were asked about how architectural decisions were handled. From their answers, it was clear that this depended on the type of decision. Small decisions, at the code level, were done by the developers. They may discuss among each other, e.g. in a code review session, and they may ask for input from the architects or others in other parts of the company. Asking for help in other parts of the company especially held true for the service layer mentioned in Section 4.2. Bigger decisions are made in the same fashion, through planned or ad hoc meetings with relevant people. The goal is to bring positive and negative sides of a decision into the light, before making the decision. None of the interviewees have reported negative experiences with this way of handling these decisions.

4.2.3 Documentation

The developers stated that they wanted more documentation on the components that they did not themselves work on, so that those could be understood. Therefore, in relation to the theme of software architecture, the interviewees were asked about documentation. The questions were oriented around three quality attributes for documentation, namely findability, maintainability, and understandability. The findings from the interviews can also be oriented around these quality attributes:

1. **Findability:** In this context, this term means how easy some document X one is looking for, is to find. The wiki has a search functionality, which is used by the developers, though interviewees said that there is an enormous amount of documentation in the wiki, which makes finding what is needed difficult. If they find a document, they are unsure if it is up to date.

   One must also note that how easy something is to find is dependent on the component. The old core system has documentation that is not available on the wiki, but in file systems which are hard to get access to. Other, newer components are using the wiki for documentation. From the interviews, it seemed that the reason for this difference was the age of the core system and that the developers on the older core system were more protective of it.

   When it comes to the project there is also some documentation on things such as user stories and decisions located in the issue tracking system.

2. **Maintainability:**

   As with the previous quality attribute, it seems that what people say about the maintainability, varies depending on the component they are primarily working on. No one is completely happy about the maintainability of the wiki, nevertheless. One developer reported that he will "...sort of give up" when there is an error in the documentation.

   The reason for this being so difficult is reported to be that there are restrictions on pages in the wiki so that you cannot easily correct an error. First, the developer has to contact the original creator, which can be difficult as the company has a lot of consultants, so the original writer often has disappeared. If the developer can be found, and he or she agrees there is an error, then he or she can correct it. To find the original author takes time. Instead, the developers create a new page with the accurate information and links to that page in a comment on the original wiki page. This problem may also present itself if the author is trying to edit their page if the project that page is associated with is considered to be completed. One of the architects said:

> *If someone has decided that we're now pretending that this project is done, so we move [all the documentation]to a locked space, and say that this documentation may be outdated. . . . And if you try to make it up to date, you do not have access. This has been a problem, and it makes, as you can imagine, a lot of outdated documentation.*

The decision to have these restrictions are made by one of the managers in the company. One of the architects said that it was brought up several times as an issue, to no avail.

3. **Understandability:** How understandable the documentation also varies, depending on whom is asked. Some of the interviewees report that the documentation is easy to understand, while others report it is impossible. It also varies with which part of the systems the documentation is for. As mentioned, the documentation for the core systems is not available in the wiki. It, of course, makes it impossible for the developers to say if it is understandable.

   There are no guidelines for what is to be documented, and no review process or quality assurance for documentation.

   The developers usually understand diagrams of the architecture which is a part of the documentation, but some report that there is missing documentation on the architecture. The diagrams are for the most part developed by one of the architects in the project, but the architecture is for the company as a whole, as described in Section 4.2.

   One of the architects, who has been in the company for some time, noted that to write documentation usually gets the lowest priority when they are developing something.

The challenges in these three areas may all be linked to the restrictions on the wiki, that most interviewees reported existing. Removing editing restrictions would remove the difficulty in finding up to date documentation, increase the ease of maintaining documentation and allow developers to improve documentation that they do not understand.

Several developers, both within the project and outside, has asked for these restrictions to be removed, but had no success.

4.3 Comparison with other Case Studies

In Chapter 2 two case studies were presented. This section will look at the case studies in relation to this one, with the goal of uncovering new sides of this case study.

4.3.1 Guidelines for Implementing Agile Practices

In Chapter 2 the case study by Rolland, Mikkelsen, and Næss (2016) was presented. The case lead to a set of five guidelines for large projects. Some of these guidelines have been followed in this project, while some have not. Following is an overview of the guidelines, with a discussion of the guidelines in the context of this case study:

1. **Improve inter-team coordination:** The goal of this guideline is to have a focus on inter-team coordination through coordination. As shown, there have not been much inter-team coordination, neither experimentation to create such arenas.

   As mentioned in Chapter 2, Rolland, Mikkelsen, and Næss (2016) recommends using long term CoPs, which according to the interviews in this case study, can be observed in the organization, but not in the project. I.e. it seems there are CoPs surrounding the project, but that they are not related to the project itself, but rather practices within the company.

   There have not been identified any use of task forces, as Rolland, Mikkelsen, and Næss (2016) also suggested.

2. **Facilitate novel practices to emerge:** As said in Chapter 2, Rolland, Mikkelsen, and Næss suggest that successful tailoring of methodologies can come from both bottom-up and top-down initiatives, but that managers should be "wary of trying to enforce predefined tailored practices".

   As mentioned, LeSS was the inspiration for the methodology used, but as shown in the previous sections, many of the It is difficult to say if the tailoring of LeSS has come from bottom-up or top-down initiatives. That being said, some of the meetings, especially those with status reporting, is an enforced practice that most participants do not enjoy.

3. **"Record, and move on":** This guideline is concerned with contractual details, i.e. they suggest that contractual details should not hinder work. That has not been seen in the case of study, and there has been seen pragmatic decisions and trust in the organization, despite its heavy reliance on external consultants.

4. **Scale the project in an evolutionary manner:** Rolland, Mikkelsen, and Næss (2016) recommends scaling the project in an evolutionary manner, so that the customer can learn the work process before the development organization is scaled. This does not seem to be the case in this project: From the interviews there are evidence that the project was prematurely scaled up, and later scaled down again. This also happened later on in the project. The tester said that at a time there were an extra tester in the project, but at that time there were not that much to test, so that tester was later removed.

5. **Adjust content in sprints:** The previous sections shows that the content of the Sprints has been adjusted, in accordance with this guideline.

This case study may be used to support the guidelines that was used, as it seems they have worked well here. As for others, such as scaling the project, there are evidence here that they have *not* scaled the project, evolutionary, and that this may be the cause of some of their challenges. Inter-team coordination has not been a focus in the project, and that guideline has therefore not been followed either. From the interviews, it does not seem that there have been a need for inter-team coordination, and that the project organization is more about gathering business and requirement resources around the product, rather than facilitating inter-team coordination. This can be seen in e.g. the meetings they have, described in Section 4.1.2.

4.3.2 Advantages and Challenges when Implementing Agile

Another case study presented in Chapter 2, was Petersen and Wohlin (2009). They did also did a case study of a large organization, and looked at benefits and issues with agile and incremental practices. Their main finding was that one cannot expect only benefits when one implements agile or incremental practices. They also identified advantages and issues with agile, and compared it with State of the Art Agile. The advantages were given in Table 2.1, and the issues in Table 2.2.

As explained in Chapter 2, it is interesting to see if the same issues and/or advantages are identified in separate case studies, so that one can tailor methods to overcome the issues that are most frequent. Based on the interviews, there have been uncovered several issues and challenges. These challenges can all be linked to two broad categories. These two categories of challenges are:

1. **Company organization and policy:** Traditional line managers and organizational elements seem to be hindering the development. This can be seen in that the APOs and PO have to spend a lot of time reporting, as do the team leads and the MPL. In the interviews, they all said that the reporting consumes time and resources that they would rather spend doing development. The reason for these meetings and all the reporting is due to demands from the company. This seems to be the case for certain technical areas as well. As one of developers put it:

   > *The problem is that we try to be an agile team, and use agile [methods]. ...But then one experiences that the company isn't an agile organization. So it creates a lot of friction, between our team and everything around us. So it creates a lot of limitations as to how agile we can be.*

2. **Methodology understanding:** It seems that some of the challenges met in the project, is due to limited understanding of artifacts in agile methods, as well as the principles from the *The Agile Manifesto*. In particular, those doing the business development, have a limited knowledge of this, as well as line managers. The reporting mentioned in the previous point is an example of this. Especially the extra meetings that are used for reporting seems to be a hindrance, as it is reported by the interviewees. The same goes for roles, explained in Section 4.1.1: The various roles are given extra responsibilities, mostly reporting about status, and approving time sheets. It was also worth noting that the teams are not cross-functional and that there are separate departments for business development. *The Agile Manifesto* states "Individuals and interactions over processes and tools....Responding to change over following a plan."

How does this relate to the issues identified by Petersen and Wohlin (2009), shown in Table 2.2? It is difficult to say, as the issues identified in Petersen and Wohlin (2009) compare results from before and after the implementation. Some of the issues can be linked to elements in this case study. The following is a description of relevant issues identified by Petersen and Wohlin (2009) and shown in Table 2.2, in the context of this case study:

CI01: According to the interviews, there are some back and forth between APOs and developers, to be able to design and implement the requirements. This happens despite meetings such as the advisory meeting and backlog grooming. Even though such meetings exsist, it takes time to decide which item is at the top of the prioritization. This can be seen in relation to the first challenge, item 1, identified above.

CI02: It is not known how "essential" the Backlog of the project is to the company, but based on the interviews it seems to be spent a lot of resources on keeping the Backlog up to date. Examples of this are the resources spent on APOs, and meetings such as the advisory meeting and Backlog grooming. It is difficult to say, based on this study alone, if the cost of keeping an Backlog up to date is higher than using other methods for handling requirements.

CI06: As mentioned regarding the production planning and go/nogo meetings described in Section 4.1.2, the number of available production windows is increasing. Nevertheless, if a delivery is not marked as a go, it has to wait for the next available production window before it can be released. The RM said that they can now order extra releases from their operations supplier, but it is not known if this has been done for the project of study. It has been mentioned in the interviews that managers have forced things into release, so that it reaches a production window. This has frustrated especially the test lead in one of the teams.

This challenge was also mapped to an issue in State of the Art methods by Petersen and Wohlin (2009).

Issues CI03-CI05 and CI07-CI12 are either not identified, or not possible to evaluate as there is not enough data on the area, e.g. because no data was gathered before agile was implemented.

In the interviews, there were suggested solutions to the challenges described above, but it may be that the interviewees are the only ones seeing the mentioned problems. I.e. managers and others do not see the problems. An example of such a solution, is to provide courses in agile methods to the employees, to increase understanding of how the developers work with agile. This can improve the productivity of the development teams, as the increased understanding can reduce pressure on the teams, and reduce time spent by team leads and APOs on reports.

When it comes to what advantages of agile that can be seen in the case, it is difficult to say, as most have said either positive or negative things about specific activities in the project, not agile has a whole. When asked about what works well in the project, almost all the developers answer that they feel that they have great colleagues. One APO said that:

> *One gets a sense of progress, and of course a sense of being more involved. It's a more fun way of working what we do now with Scrum, as opposed to when with did a large projection up front.*

The same APO also said that both the two dedicated teams, and how they work with the APOs, works well, when asked what works well in the project. Developers are also happy about how they deliver, and feel that they deliver. One developer said:

> *I think a lot of things works well. There has been a lot of challenges, . . . but I think we are good at turning around at deliver, or work effectively. I think what we deliver is great.*

Looking at Table 2.1, the following advantages can be identified in the this case study, based on the interviews:

CA04: Ad hoc meetings and direct communication is observed in the case. Interviewees report that it is easy to get in contact with those in the team. That being said, there is still a lot of documentation which they must spend time understanding. This is because they have to work with components that are developed in other parts of the company. It is also worth mentioning that the teams are not that small. One of them has 9 members, while the other has 8 members.

CA07: There are short ways of communication between testing, developers and APOs which seems to works well. There are challenges in the area, nevertheless: One of the test leads reports that he feels that epics are not specified enough when they come for testing, or that they are not ready in time for them to get a good understanding of the epics. The test lead reported that this makes testing more difficult.

CA01-CA03, CA05-CA06 and CA08 are either not identified, or it is not possible to evaluate if they are in the case. The reason for this is that there was not conducted interviews before agile methods was introduced in the organization. The advantages reported by Petersen and Wohlin (2009) seems to see improvements in metrics, which are difficult to spot here because of the approach to time in the study. This is described in Section 3.2.2.

Most of the advantages base themselves on small teams. This project has had one team with 9 members, while the other team had 8 members.

4.4 Research Summary

This chapter has presented various aspects of the case study and explored numerous challenges within these aspects, and how they relate to both LeSS and other case studies, presented in Chapter 2. In summary, they have done multiple tailorings of LeSS. Most notable are the following:

1. A Project Management Office (PMO) that controls the project, which in turn is controlled by a "control group" in the company.
2. A go/ngo-meeting to determine if a task is ready for production.
3. A "business status meeting" where APOs and team leads repeat their written report to the PMO.
4. Common activities with both teams have been removed.
5. An advisory meeting as a meeting for the architects to give APOs feedback on suggested epics and requirements.

There have been identified two main benefits of working as they do: A sense of progress for the involved, and improved communication between the business end of the company and the developers.

There have also been challenges, and these can be linked to either lack of understanding of the methodology in the organization, or to organizational policies that are out of the projects control.

4.5 Research Significance

For the results of this master thesis to be significant, one must be able to use them for something. How can these results be used in other projects or organizations, which are trying to implement agile methodologies?

To mitigate or overcome the two main categories of challenges explained in Section 4.4, one must first be aware that those are possible challenges. By showing these in one organization, one can easier find them in other organizations. When other organizations are trying to implement agile, they can ensure that their employees are educated on the methodologies. By being aware of the challenges related to how agile methods are combined with their organizational structure, one can adjust the organization in advance of introducing agile methods. One can also use these findings when developing agile methods, such as LeSS.

Chapter 5

Conclusion

The previous chapter presented and discussed the findings from the 21 interviews that were conducted, to explore how a large company has used agile methods and tailored the methods to their needs. This chapter will summarize the findings, and show how the research questions from Chapter 1 have been answered. The main question was:

> *How can a large organization use agile methodologies in a development project with multiple teams, so that the teams can develop systems that serve the needs of multiple stakeholders within the organization?*

Which was broken down into two sub-questions:

RQ1: How is the development project organized in terms of teams, roles, meetings and team activities?

RQ2: How has the organization tailored a agile methodology, so that it fits the organizations enterprise needs?

Section 5.1 will outline the answer to RQ1, while Section 5.2 will outline the answer to RQ2.

5.1 Agile Elements in a Large Project

RQ1 has been answered in Section 4.1.2 and Section 4.1.1. The way it is organized can be summarized as follows: It has a Product Owner (PO) who has several Area Product Owners (APOs) which define requirements. These APOs may use analysts to develop the requirements. Then they pitch these in an advisory meeting, where architects can say how much time the requirement will take to fulfill, outline a solution or tell them that it is not possible to do. When the architects are happy with the requirements, they can be prioritized. The prioritized tasks are allocated between the two development teams. They use meetings such as standups, Backlog grooming and sprint planning to organize their work. When a team is done with their work, they have a demonstration where stakeholders in the company can give feedback, as well as a retrospective. All of these activities are done within each team.

When it comes to architecture, they use a model similar to the one defined by Eckstein (2014) as the "Chief architect": There is one chief architect, but there is an additional architect one that sometimes assists in the work, e.g. in meetings. The chief architect works by being available to the developers for questions, which the person does by wandering around the developers, and sitting with them. The model works for the project, but it seems that the chief architect has a lot to do, and may be overworked.

5.2 Tailoring Large-Scale Scrum

RQ2 is answered throughout Chapter 4: The project used the elements from Large-Scale Scrum (LeSS), with tailoring. Of the tailorings, the following are the most notable ones:

1. A Project Management Office (PMO) that controls the project, which in turn is controlled by a "control group" in the company.
2. A go/nogo-meeting to determine if a task is ready for production.
3. A "business status meeting" where APOs and team leads repeat their written report to the Project Management Office (PMO).
4. Common activities with both teams have been removed.

5. An advisory meeting as a meeting for the architects to give APOs feedback on suggested epics and requirements.

Some tailorings was due to enterprise needs of the organization, i.e. items 1-3 above. These tailorings seemed to be associated linked to various challenges for those that were interviewed. Most these challenges could be linked to one of two things:

1. A traditional organization surrounding the project, which meant that traditional project management activities such as the above mentioned reporting had to be done. This consumed time for those working on the project.
2. A limited understanding of agile in the parts of the company and project which have not worked agile before. The reporting is an example of this. Knowing how one sees results in an agile project, would help managers in the company to not demand such reports, which in turn would benefit those working on the project.

The most promising of the above tailorings is the advisory meeting (item 5 above), where the business developers or APOs pitch their ideas or needs for development to architects. This enables the architects to control the direction of the systems, while the business developers can receive advice about resources and areas in which the requirement or idea is unclear.

They also chose not to include some elements from LeSS, most notably the Overall Retrospective and the common Sprint Planning One. This meant that in the project, there were none common activities for both the teams.

5.3 Research Limitations

Chapter 4 has presented what was found through the research method explained in Chapter 3. The findings are summarized in Section 4.4. Due to how the research was conducted, there are some limitations to the findings, that should be noted:

1. **Number of interviews:**

Table 3.1, Table 3.2 and Table 3.3 shows how many people of the different roles has been interviewed in this study. As the reader can see, everyone has not been interviewed. More specifically, there are nine people in team 1, while six were interviewed. Team 2 has eight members, where four members were interviewed. As for the project broad roles, there were 20 people, while 11 were interviewed. The fact that there is a limitation in the number of interviews is a conscious decision to be able to meet time restrictions. This time restriction is also mentioned in Section 3.3, as more interviews were allowed by the people responsible for the projects.

Nevertheless, the limited number of interviews is a limitation of the findings, as those not interviewed could give contradictory statements to those collected in this study, or given a nuanced description of the situation in various areas.

2. **Limited parts of the organization interviewed:**

   A common theme in the findings is a criticism of the organization, or other parts of the organization, which has made decisions which affect the project. A limitation in these findings is that those making those decisions have not been given a chance to be interviewed, which makes these findings one-sided.

3. **Number of data generation methods:**

   As described in Chapter 3, there was only one data generation method: The semi-structured interviews. As the author was given full access to the project, one could also have included artifacts such as documentation in the analysis. One could also have conducted surveys, or used other data generation methods for associated parts of the organization, giving a more holistic view of the enterprise. The reason for this not being done, as with previous limitations, due to time constraints.

4. **Combination of methodologies:**

   They have not used only one methodology, which makes findings regarding the methodologies themselves limited. The findings are limited to a tailoring and combination of the methods, such as the one in the case. Similar organizations and projects are the ones who can draw from this, not the methodologies.

5. **The limitations of the researcher:**

   As it was described in Chapter 3, an inductive approach for creating the categories for the data analysis was selected. The interview guide and the interviews themselves were also conducted inductively. The researcher was in complete control over all parts of the research process. This means that the research findings are prone to any faults that the researcher may or may not have made.

Because of these limitations, the results should be read with some degree of skepticism. That being said, the study was an exploratory one, and the limitations could be cleared through further research into the company and project. The next chapter, Chapter 5 will explore what conclusions can be drawn based on the findings presented in this chapter, and point a direction for further research into this subject.

5.4 Future Work

The findings in this research have determined that there are challenges when one tries to implement agile methods in a large company. It also opens for new areas in which one should investigate. The same data set, together with additional interviews, can be used to explore further:

- Other areas in the company which try to work agile, and see how that compares to this project. Is it easier to be agile in other projects in the same company?
- The relation between the teams located in Norway, and the one in Bratislava, and how the various part of the relationship feel about it. This could be used to explore the use of remote teams in Norwegian development projects.
- As the organization never tried out the LeSS Framework as it is described by its creators, it would also be interesting to see how successful an implementation of the framework would be in the same organization.

Because of the limitations, in this case, one should also further investigate a project which has completely committed itself to LeSS. When it comes to Scrum, Kanban, and LeSS, the findings here are limited as they are combined. Comparing such results with this research could give insight into if tailoring methods is a good idea in the first place. Will using defined methodologies produce better results?

It would also be interesting to explore a project with more teams that are working together on a product. LeSS is recommended for 2-8 teams, which makes this an exploration of the lower limit.

References

Fowler, Martin and Highsmith, Jim. *The Agile Manifesto*. URL: http://agilemanifesto.org/ (visited on 06/22/2017).

Sutherland, J. (2014). *Scrum: The Art of Doing Twice the Work in Half the Time*. 1st ed. Crown Business.

Beck, K. and C., Andres (2004). *Extreme Programming Explained: Embrace Change*. 2nd ed. Addison Wesley.

VersionOne (2016). *VersionOne's 11th Annual State of Agile Report*. URL: https://explore.versionone.com/state-of-agile/versionone-11th-annual-state-of-agile-report-2 (visited on 06/22/2017).

Nord, R. L., Ozkaya, I., and Kruchten, P. (2014). "Agile in Distress: Architecture to the Rescue". In: *Agile Methods: Large-Scale Development, Refactoring, Testing, and Estimation*. Ed. by Dingsoyr, T. et al. Vol. 199. Lecture Notes in Business Information Processing. Berlin: Springer-Verlag Berlin, pp. 43–57. ISBN: 978-3-319-14358-3; 978-3-319-14357-6. URL: <GotoISI>://WOS:000357375600005.

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

Dingsøyr, T. et al. (2016). *Exploring Software Development at the Very Large-Scale: An exploratory case study and research agenda for agile method adaptation*.

Rodriguez, P. et al. (2012). *Survey on Agile and Lean Usage in Finnish Software Industry*. Conference Paper. DOI: 10.1145/2372251.2372275.

Dingsøyr, Torgeir et al. (2012). "A decade of agile methodologies: Towards explaining agile software development". In: *Journal of Systems and Software* 85.6. Special Issue: Agile Development, pp. 1213 –1221. ISSN: 0164-1212. DOI: http://dx.doi.org/10.1016/j.jss.2012.02.033. URL: http://www.sciencedirect.com/science/article/pii/S0164121212000532.

Ries, E. (2012). *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. 1st ed. Crown Business.

Petersen, Kai and Wohlin, Claes (2009). “A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case”. In: *Journal of Systems and Software* 82.9, pp. 1479 –1490. ISSN: 0164-1212.

Hobbs, B. and Petit, Y. (2017). “Agile Methods on Large Projects in Large Organizations”. In: *Project Management Journal* 48.3, pp. 3 –19. URL: https://www.pmi.org/learning/library/agile-methods-large-organizations-projects-10767.

Dingsoyr, T. and Moe, N. B. (2014). “Towards Principles of Large-Scale Agile Development A Summary of the Workshop at XP2014 and a Revised Research Agenda”. In: *Agile Methods: Large-Scale Development, Refactoring, Testing, and Estimation*. Ed. by Dingsoyr, T. et al. Vol. 199. Lecture Notes in Business Information Processing. Berlin: Springer-Verlag Berlin, pp. 1–8. ISBN: 978-3-319-14358-3; 978-3-319-14357-6. URL: <GotoISI>://WOS:000357375600001.

VersionOne (2015). *VersionOne’s 10th Annual State of Agile Report*. URL: https://explore.versionone.com/state-of-agile/versionone-10th-annual-state-of-agile-report-2 (visited on 07/05/2017).

Highsmith, J. and Cockburn, A. (2001). “Agile software development: the business of innovation”. In: *Computer* 34.9, pp. 120–127. ISSN: 0018-9162. DOI: 10.1109/2.947100.

Wall, Matthew (2014). *Innovate or die: The stark message for big business*. BBC News. URL: http://www.bbc.com/news/business-28865268 (visited on 07/04/2017).

Söderström, Jonas (2013). *Jævla drittsystem!* Spartacus Forlag AS.

Sugimori, Y. et al. (1977). “Toyota production system and Kanban system Materialization of just-in-time and respect-for-human system”. In: *International Journal of Production Research* 15.6, pp. 553–564. DOI: 10.1080/00207547708943149. eprint: http://dx.doi.org/10.1080/00207547708943149. URL: http://dx.doi.org/10.1080/00207547708943149.

Ahmad, M. O., Markkula, J., and Oivo, M. (2013). “Kanban in software development: A systematic literature review”. In: *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 9–16. DOI: 10.1109/SEAA.2013.28.

Brooks, Frederick (1975). *The Mythical Man-Month*. Addison Wesley.

Larman, C. and Vodde, B. (2016). *Large-Scale Scrum: More with LeSS*. 1st ed. Addison-Wesley.

LeSS Framework. URL: https://less.works/less/framework/index.html (visited on 06/26/2017).

Buchmann, Felix, Nord, Robert L, and Ozakaya, Ipek (2012). *Architectural Tactics to support rapid and agile stability*. Tech. rep. DTIC Document.

Rolland, Knut H., Mikkelsen, Vidar, and Næss, Alexander (2016). "Tailoring Agile in the Large: Experience and Reflections from a Large-Scale Agile Software Development Project". In: *Agile Processes, in Software Engineering, and Extreme Programming: 17th International Conference, XP 2016, Edinburgh, UK, May 24-27, 2016, Proceedings*. Ed. by Sharp, Helen and Hall, Tracy. Springer International Publishing, pp. 244–251. ISBN: 978-3-319-33515-5.

Dingsøyr, Torgeir et al. (2017). "Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation". In: *Empirical Software Engineering*. ISSN: 1573-7616.

Bass, L., Clements, P., and Kazman, R. (2012). *Software Architecture in Practice*. 3rd ed. Addison Wesley.

Svorstøl, Stein-Otto (2016). "The role of architecture in large-scal eagile projects". In:

Eckstein, J. (2014). "Architecture in Large Scale Agile Development". In: *Agile Methods: Large-Scale Development, Refactoring, Testing, and Estimation*. Ed. by Dingsoyr, T. et al. Vol. 199. Lecture Notes in Business Information Processing. Berlin: Springer-Verlag Berlin, pp. 21–29. ISBN: 978-3-319-14358-3; 978-3-319-14357-6. URL: <GotoISI>://WOS:000357375600003.

Larman, C. and Vodde, B. (2010). *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum*. 1st ed. Addison-Wesley.

Eckstein, J. (2004). *Agile Software Development in the Large: Diving into the deep*. Dorset House Publishing Company Inc.

Kruchten, P. (2013). "Contextualizing agile software development". In: *Journal of Software-Evolution and Process* 25.4, pp. 351–361. ISSN: 2047-7481. DOI: 10.1002/smr.572. URL: <GotoISI>://WOS:000318030600004http://onlinelibrary.wiley.com/store/10.1002/smr.572/asset/smr572.pdf?v=1&t=iue0na04&s=8abecf55190766711288667eaeaf565dab75472c.

Oates, B.J. (2006). *Researching Information Systems and Computing*. 1st ed. SAGE Publications Inc.

Ltd., QSR International Pty (2017). *What is NVivo?* URL: http://www.qsrinternational.com/what-is-nvivo (visited on 06/22/2017).

Atlassian. *Working With Epics*. URL: https://confluence.atlassian.com/agile/jira-agile-user-s-guide/working-with-epics (visited on 06/26/2017).

Glossary

epic "An epic captures a large body of work. It is essentially a large user story that can be broken down into a number of smaller stories. It may take several sprints to complete an epic."(*Working With Epics*) . v, vi, 51–57, 59–65, 76, 81

Backlog Refers to an accumulation or list of unfinished work or unfilled orders. In software development methodologies, such as LeSS and Scrum, it is an ordered list of unfinished work, with the most important items at the top. (Sutherland, 2014; Larman and Vodde, 2016). v, 15–18, 51, 53, 55, 56, 64, 74, 80

Sprint A time-box of some length, usually 30 days or less, in which development activities are conducted. The purpose of the time-box is managing stakeholder expectations, by committing to a certain amount of work for the time-box. (Sutherland, 2014). 16–20, 55–58, 60, 72, 81, 89

Sprint Backlog In some software development metholdogies, such as LeSS and Scrum, it refers to the list of items that a team has comitted to for the next Sprint. (Sutherland, 2014; Larman and Vodde, 2016). 16, 18

Definition of Done A definition of what it means for an item or task to be "done" in a project using the Scrum or LeSS methodologies. This enables the teams to measure completion binary: Either a task is done, or not done, and one can see that e.g. 15 tasks are not done, while 10 are. (Larman and Vodde, 2016, p. 229). 16, 64

fullstack A collective term for both the frontend and backend.. 23

feature team A long-lived, cross-functional, cross-component team that completes many end-to-end- customer features one-by-one.Larman and Vodde, 2010, p. 549. 24–27

user story “A specification of one type of interaction with a system.”(Sommerville, 2011, p. 747). 53, 55, 56, 64, 65, 69

layer A logical structuring of software elements. The term is associated with the multi-layer/n-layer archtectural pattern.. 66–68

Service-Oriented Architecture An architectural pattern in which discreet units of functionality are offered through self-contained components. . 66

frontend Refers to the presentation layer/User Experience (UX), what the user sees, in a computer system.. 89

backend Refers to the business logic and data handling, what the user does not see, in a computer system.. 89

Acronyms

NTNU Norwegian University of Science and Technology. 3, 8

LeSS Large-Scale Scrum. v, 8, 9, 11, 13–19, 25, 49–63, 71, 72, 76, 77, 80, 81, 83, 84, 89

PMO Project Management Office. v, 50, 51, 55, 59, 61, 63, 64, 76, 80

APO Area Product Owner. v, vi, 17, 50–57, 59–65, 73–76, 80, 81

UX User Experience. vi, 23, 52, 57, 58, 61, 90

XP Extreme Programming. 3, 9, 21

LSU Lean Startup. 3, 9

TTM Time To Market. 3, 6

ECTS European Credit Transfer and Accumulation System. 8

PO Product Owner. 15–19, 27, 51, 53, 57, 59, 60, 73, 80

PBR Product Backlog Refinement. 16, 18, 55

CoP Community of Practice. 20, 25–27, 62, 67, 71

LSV Latest System Version. 28, 29

TST Technical Service Team. 27

TCT Technical Consulting Team. 27

MPL Main Project Lead. 52, 57, 60, 64, 73

RM Release Manager. 53, 62, 63, 74

Appendices

Appendix A

Interview guide 1

The interview guide follows on the next page.

Before we begin

The interviewer informs the interviewee

- Role and background
  - NTNU, project on large-scale agile with 170+ developers
- Master thesis with case study on large-scale methods
- Came in contact with the Company, which has multiple teams for a project.
- Interviews
  - Will be recorded, then transcribed, but all people, the project and the Company, will be anonymized
  - Answer however you like, no right or wrong answers. Explain as much as you like. May ask more questions, but I want you to talk the most.

Introduction

Goal: Get to know the interviewee, and get a picture of their role and work.

- Internal, or external?
- Time on the project?
- What is your role in the project?
- What are your tasks? Responsibilities?
- What is, in your own words, the purpose of the project?
- Describe how the project and your team is organized?
- Who are the main stakeholders in this project?

Methodology

Goal: Find out which software development methodology the interviewee believes the project is using, how it's meant to work (if that is something that's communicated to the interviewee), how it works in practice in this project, and how the people involved relate to it.

- How would you define "software development methodology"?
- How would you describe the software development methodology on this project?
- In the previous week, which meetings and/or activities did you take part in? (Can describe week etc.)
  - Who were at the mentioned meetings/activities?
  - What is the purpose of the activity/meeting?
  - What was your role there?
  - What do you think about the meeting/activity? / To what extent do you find the meeting/activity useful to you in your work/role?
- Are there any other arenas (than those mentioned in the previous question) where you communicate within your team, or with other teams?
- What works well on this project?
- What is challenging on this project?
- If not already answered through the previous questions:

    - How is work assigned?
    - What’s the timeline for assigned work?
    - Who do you contact in case of questions about how the product/work should function?
    - Who do you contact in case of technical problems?
    - How/when/where are dependencies coordinated?

Software Architecture

Goal: Find out how the interviewee sees architecture in the project, how it’s meant to work, how it works on the project and how the people involved work with it.

First, if the interviewee has a non-technical role:

- Who are the architects in this project?
- How would you describe their role?
- How do you work with/relate to the architects

Everyone (if previous section meant that the person had no relation to the architecture, or architectural work, this may be skipped):

- How do you define “software architecture”?
- How would you describe the overall software architecture of your project?
- Who are the architects in this project?
- How would you describe their role?
- When you have a question regarding the architecture, who do you contact?
- How is the architecture documented?
    - What in the project is documented, and where is the documentation?
    - How would you describe the documentation in terms of:
        - Discoverability/findability
        - Understandability
        - Maintainability
- How is the architecture and architectural decisions communicated to the teams?
- How are decisions made, and who makes them? Outside the team?
- How are trade-offs handled?
- Has there been changes to the architecture over time? What are the reasons for these changes?
    - How are change requests from stakeholders that demand architectural change handled?

Ending the interview

- Comments?
- Questions?
- Inform the interviewee about my Contact information
    - Within the company.
    - At the university

Appendix B

Interview guide 2

The interview guide follows on the next page.

Before we begin

The interviewer informs the interviewee

- Role and background
    - NTNU, project on large-scale agile with 170+ developers
- Master thesis with case study on large-scale methods
- Came in contact with the Company, which has multiple teams for a project.
- Interviews
    - Will be recorded, then transcribed, but all people, the project and the Company, will be anonymized
    - Answer however you like, no right or wrong answers. Explain as much as you like. May ask more questions, but I want you to talk the most.

Interview questions

Goal: Get to know the interviewee, get a picture of their role and work and how the person works with the development teams

- Internal, or external?
- Time on the project?
- What is your role in <PROJECTNAME>?
- What are your tasks and/or responsibilities?
- What is, in your own words, the purpose of the project?
- Describe how the project, you, your coworkers and the development teams are organized?
- Which meetings or activities do you take part in, that are related to <PROJECTNAME>?
    - How and when do you interact with the developers?
    - Are you familiar with the term “software development methodology”?
        - What does it mean to you?
        - What’s the software development methodology employed in <PROJECTNAME>
    - Are there any other arenas where you communicate with the development teams, or with other relevant stakeholders?
- What works well on this project?
- What is challenging on this project?

Ending the interview

- Comments?
- Questions?
- Inform the interviewee about my Contact information
    - Within the company.
    - At the university

Appendix C

E-mail sent with transcribed interview to interviewees

Subject: Transkribert intervju til gjennomlesning

Body:

Hei,

nå har jeg transkribert intervjuet jeg gjorde med deg i forbindelse med masteroppgaven min. Det er vedlagt, slik at du kan lese gjennom om du ønsker, og gi beskjed om ting du eventuelt ønsker å endre, eller som du ønsker å legge til. Filen er kryptert, i tilfelle den skulle komme på avveie. <Password hint>.

Transkriberingen er på et format hvor spørsmål eller ting jeg sier er i fet skrift, og det du sier er indentert og i vanlig skrift. Når noe er sagt, men jeg ikke har hørt hva det er, er det markert med doble, tomme parenteser: (()). Om jeg har hørt noe, er litt usikker på hva som ble sagt men har en idé om hva det kan være, så er det markert med doble parenteser med det jeg tror jeg hørte inn: ((Det jeg tror ble sagt)). Hendelser er representert i kommentarer, som er formatert i kursiv med enkle parenteser rundt: (*En kommentar*). Lydord er forsøksvis filtrert ut.

Gi beskjed så snart som mulig om det er noe du vil endre, eller noe du vil legge til.

Dersom du lurer på noe kan du sende meg en e-post, eller ringe meg på <phone number>.

Takk igjen for bidraget!

Vennlig hilsen Stein-Otto Svorstøl <Contact info within the company>