UNIVERSITETET I AGDER

Challenges with Agile in a System Development Department: A Case Study

MARIUS ANDERSEN BJØRNI

SIMEN HAUGEN

For the Master's Degree in

Industrial Economics and Technology Management

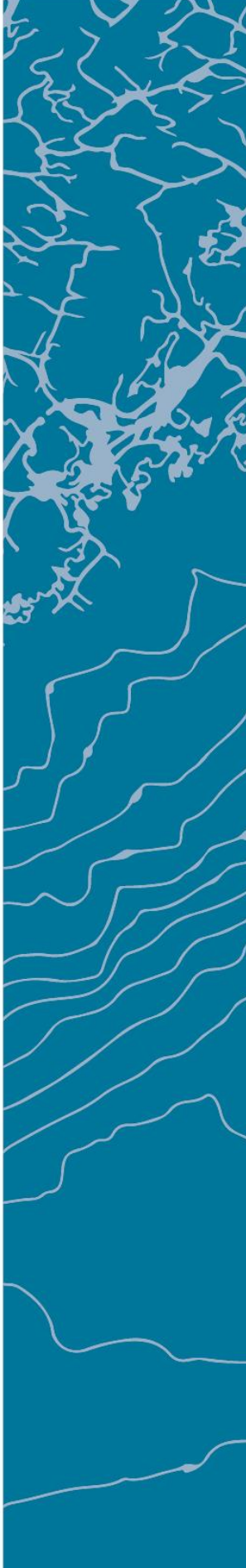
SUPERVISOR

Knut Erik Bonnier

University of Agder, 2019

Faculty of Engineering and Science

School of Business and Law



UiA
University of Agder
Master's thesis

Faculty of Engineering and Science
School of Business and Law

Abstract

Even though agile approaches are renowned for rapid and efficient adaption to market changes, decreased time for solving client demands, and more value for the customer, they may also introduce unfortunate challenges that can hinder the productivity of organizations. It is challenging to implement and use agile approaches in organizations successfully, and while well-known challenges are well understood, new ones are emerging. This master thesis aims to uncover the potential challenges with agile in a system development department through a qualitative, descriptive case study, where interviews of key personnel and observations were conducted. Nine challenges with associated causes and consequences were uncovered, most of them already prevalent in existing theory. The case study did, however, uncover a challenge related to the estimation of maintenance work in upcoming Sprints. This challenge is not covered in the Scrum framework and is therefore suggested as a subject for future research.

Preface

This master thesis serves as the concluding part to the Master's Program in Industrial Economics and Technology Management at the University of Agder.

We wish to thank the case company for giving us the opportunity to write this thesis, their hospitality, and all the employees who took the time to let us interview them. None of this would be possible without our acquaintance who facilitated the contact with the case company, so we are very grateful to him.

We would also like to thank our supervisor for providing feedback during the writing process. Finally, thanks to everyone who has supported us this semester and helped us finish our thesis.

Grimstad, May 2019

Marius Andersen Bjørni

Simen Haugen

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Agile has, since its formulation in the agile manifesto back in 2001, lead to unprecedented changes in the software development field, introducing several different methods and practices (Dingsøyr, Nerur, Balijepally, & Moe, 2012). It has also spread to a wide range of other fields and functions and is now used in everything from human resources to marketing to production of fighter jets (Rigby, Sutherland, & Takeuchi, 2016). Furthermore, according to the Annual Report by World Economic Forum (2018), their team of over 700 collaborators around the world has remained conscious towards their long-term mission "to create lasting yet agile structures that can respond to the ever-changing and challenging environment that affects all stakeholders in the quest for a positive future.".

Even though agile approaches are renowned for rapid and efficient adaption to market changes, decreased time for solving client demands, and more value for the customer (Stoica, Mircea, & Ghilic-Micu, 2013), they may also introduce unfortunate challenges that can hinder the productivity of organizations. It is challenging to implement and use agile approaches in organizations successfully, and while certain challenges are well understood, new ones are emerging (Gregory, Barroca, Sharp, Deshpande, & Taylor, 2016). To get a better understanding and overall implementation of agile, emerging challenges need to be identified and analyzed.

1.2 Problem Statement

This master thesis aims to uncover challenges that may arise when implementing agile development approaches. The problem statement of this thesis is as follows:

- What are the potential challenges with agile in a system development department?

1.2.1 Research Questions

The problem statement has been divided into two research questions to specify further what the focus of this thesis is, and how the problem statement will be answered.

- **RQ: 1** What challenges related to agile and its implementation exist in the system development department at the case company?
- **RQ: 2** What are the underlying reasons and consequences for these challenges?

In addition to answering the two research questions, potential solutions or mitigation advice is given for each challenge.

1.3 Delimitations

Given the size and structure of the system development department of the case company (which is more thoroughly explained and described in section 4.2), the case study of this thesis will only focus on the internal subdivisions of the system development department. The reason for this is because the external subdivisions do not necessarily follow the same agile approaches as the internal subdivision.

While the system development department is an integrated part of a larger company, this thesis only focuses on the challenges they experience. There

might be more challenges in the other parts of the organizations that relate to the agile implementation, but they are out of scope for this thesis.

Challenges not related to agile or its implementation in the system development department are out of the scope of this thesis. These challenges are, therefore, excluded. The criteria for excluding challenges are that the challenges were present before the agile transformation in the system development department and that they have not been affected by the agile transformation or the current agile work methods.

1.4 Thesis Outline

This master thesis is divided into six chapters. The first chapter gives an introduction to the topic and problem statement of the thesis. The second chapter outlines the theory used to analyze the data gathered from the case study. Chapter three explains the research methods used. The fourth chapter gives a brief description of the case company, including how the system development department of the organization is structured, its work methods, and the different roles of its employees. Chapter five presents the findings from the data gathering and discusses them in relation to existing theory. Finally, chapter six concludes the thesis and highlight how this study contributes to theory and practice and indicate directions for future research.

Chapter 2

Theory

This chapter gives an introduction to the primary fields of research for this thesis. It contains project management, two subcategories: traditional and agile software development, the agile framework Scrum, and lastly challenges of agile implementation.

2.1 Project Management

Project management can be defined as "The application of knowledge, skills, tools, and techniques to project activities to meet the project requirements." (Larson & Gray, 2010, p. 647). IT projects can pose a serious threat to companies, as they more often than other projects end up far exceeding their schedules and budgets (Flyvbjerg & Budzier, 2011). A study by Bloch, Blumberg, and Laartz (2012) found that large IT projects on average run 45 % over budget, 7 % over schedule and underdeliver value by 56 %. High uncertainty, constantly changing business and user requirements, and evolving business environment are among the reasons why managing IT projects is a complex and difficult task (Rahmanian, 2014).

2.2 Traditional Software Development

Traditional software development is, amongst more, characterized as being: process-centric, command and control managed, guided by tasks or activities, following life cycle models (like Waterfall), requiring substantial documentation and conducting all planning up-front (Conboy, Coyle, Wang, & Pikkarainen, 2011). The fundamental assumption it is based upon is according to Dybå and Dingsøyr (2008) that systems can be fully predicted and specified, and that meticulous and extensive planning is part of building them. The traditional approach to software development which "aim to address the whole software project lifecycle, e.g., by providing comprehensive guidelines, standardized procedures, project planning templates, and interfaces to further organization processes" (Theocharis, Kuhrmann, Münch, & Diebold, 2015, p. 150), has been found to result in excessive rework, inflexibility, customer dissatisfaction, and sometimes to be outdated by time of completion (Serrador & Pinto, 2015).

2.3 Agile Software Development

According to Agile Alliance, agile software development is "an umbrella term for a set of frameworks and practices based on the values and principles expressed in the Manifesto for Agile Software Development and the 12 principles behind it." (Agile Alliance, n.d.-b). Agile software development highly emphasizes the importance of having self-organizing and cross-functional teams that work closely with the customer and end users, as well as continuously delivering functioning and valuable software (Agile Alliance, n.d.-b). The term "agile software development" dates back to 2001 when a group of 17 people, each with their way of practicing software development, got together at a ski resort in Snowbird, Utah to figure out the commonalities between their way of developing software. The result of this meeting was the Manifesto for Agile Software development, a collection of a set of four values and 12 principles (Agile Alliance, n.d.-b).

The Manifesto for Agile Software Development contains the following four values (Beck et al., 2001):

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

The Manifesto for Agile Software Development also states that the items on the left are valued more than the ones on the right (Beck et al., 2001). Furthermore, the Manifesto for Agile Software Development has also defined 12 principles (Beck et al., 2001):

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and the support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversations.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity - the art of maximizing the amount of work done - is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how they become more effective, then tunes and adjusts its behavior accordingly.

2.4 Scrum

Schwaber and Sutherland (2017)[1] defines Scrum as "A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value.". The framework was developed by Ken Schwaber and Jeff Sutherland in the early 1990s (Schwaber & Sutherland, 2017) and has since been widely adopted in software development. The Scrum framework is illustrated in figure 2.1.

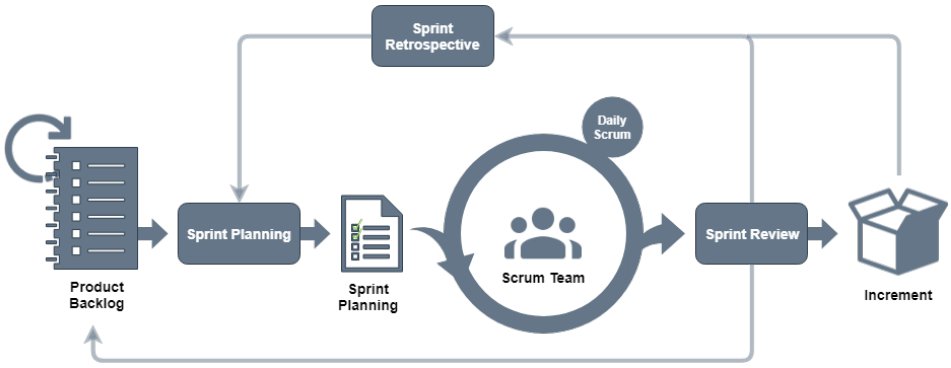


Figure 2.1: The Scrum framework (Adapted from Scrum.org (n.d.)).

The Scrum framework is made up of Scrum Teams and their affiliated roles, events, artifacts, and rules. These various components within the framework each serve a specific purpose and are fundamental to achieve an optimal Scrum implementation (Schwaber & Sutherland, 2017).

[1]The majority of the Scrum theory used in this thesis is derived from one source, Schwaber and Sutherland (2017), as it is written by the founders of the Scrum framework.

2.4.1 The Scrum Team

The Scrum framework highly emphasizes the importance of having self-organizing and cross-functional Scrum Teams. Self-organizing teams decide on their own how their work should be carried out and accomplished, as opposed to being instructed by people outside the team. Cross-functional teams have the advantage of possessing a variety of knowledge required to complete the given work and not being dependent on other parties outside the team. The Scrum Team includes a Product Owner, the Development Team, and a Scrum Master (Schwaber & Sutherland, 2017).

Product Owner

The Scrum Guide states that the overall objective and responsibility of the Product Owner is to maximize the value of the product resulting from the work of the Development Team (Schwaber & Sutherland, 2017). However, how this is implemented may vary between organizations and Scrum Teams. The Product Owner accomplishes this work through what is often referred to as Product Backlog management. Product Backlog management entails arranging the items in the Product Backlog (see section 2.4.3) in a prioritized order, making sure the Product Backlog is visible, transparent and understood by everyone on the Development Team, as well as showing what the Scrum Team will be working on the following Sprint (Schwaber & Sutherland, 2017).

Development Team

The Development Team consists of the dedicated professionals within the Scrum Team who executes the prioritized items in the Product Backlog (Schwaber & Sutherland, 2017). As mentioned above, the Development Team is both self-organizing and cross-functional, which enhances the Development Team's overall efficiency and effectiveness. Furthermore, the Scrum framework does not recognize titles for members of the Development Team, nor does it recognize subteams within the Development Team. Even though individual members of the Development Team possess specialized skills or have areas of focus, accountability for work done still belongs

to the Development Team as a whole. The number of people to be in the Development Team varies across different organizations and Scrum Teams, but the optimal number is considered to be between three and nine people. Having more than nine team members often lead to difficulties related to coordination, and having less than three team members may lead to skill constraints and reduced interaction during the Sprint, which may lead to smaller productivity gains (Schwaber & Sutherland, 2017).

Scrum Master

The Scrum Master's primary responsibility is to ensure that all the various parts of the Scrum process and theory are followed throughout the Scrum Team and work as a whole (Schwaber, 2000, p. 36). This responsibility entails teaching other members of the Scrum Team how to use the Scrum process to handle problems that arise during the project, as well as to help other people of the organization better understand how to interact with the Scrum Team to maximize the value they create (Schwaber & Sutherland, 2017). The Scrum Master's responsibility towards the Development Team is among other things to remove any impediments that may negatively affect their progress, to facilitate creativity and empowerment (Schwaber, 2000, p. 36), and to arrange Scrum Events when necessary or requested (Schwaber & Sutherland, 2017). The Scrum Master's service to the Product Owner is to advise how he or she can maximize return on investment to meet the goals of the organization or project through Scrum (Schwaber & Sutherland, 2017).

2.4.2 Scrum Events

The Scrum framework consists of several different events in order to sustain regularity and to reduce the need for meetings that are not part of Scrum (Schwaber & Sutherland, 2017). Although the various events serve different purposes, they all share a common rule. All events within the framework are time-box events, meaning that every type of event has a maximum duration (Schwaber & Sutherland, 2017). The various events are more thoroughly explained in the following paragraphs.

Sprint

Schwaber and Sutherland (2017) defines a Sprint as "A time-box of one month or less during which a 'Done'[2], usable, and potentially releasable product Increment is created.". An Increment within the Scrum framework is defined as the total amount of all the Product Backlog Items (PBIs) that have been completed during a given Sprint (Schwaber & Sutherland, 2017).

Each Sprint may be described as a project that spans for maximum one month. Like a project, each Sprint has a goal of what is to be achieved or developed, a design and flexible plan that describes how this will be achieved, as well as the finished product at the end. A new Sprint cannot start until the previous has been concluded (Schwaber & Sutherland, 2017).

The reason why Sprints are limited to one month is related to the potential challenges that may arise in Sprints with longer time-boxes, such as changes to the definition of what is being developed or an increase in complexity or risk. By limiting a Sprint's time-box, one can enable predictability by ensuring inspection and adjustment of progress toward the Sprint Goal[3] at least every 30 days (Schwaber & Sutherland, 2017).

A Sprint is made up of the Sprint Planning, Daily Scrums, the work related to development, Sprint Review, and Sprint Retrospective (Schwaber & Sutherland, 2017).

Sprint Planning

The work to be accomplished during the upcoming Sprint is planned at the Sprint Planning meeting. The length of a Sprint Planning may vary, however, Schwaber and Sutherland (2017) state that the meeting should not exceed eight hours for a one-month Sprint. It is the Scrum Master's responsibility to make sure the event takes place. Sprint Planning aims to answer the following two questions (Schwaber & Sutherland, 2017):

[2]According to Schwaber and Sutherland (2017), the definition of 'Done' is a shared understanding between members of the Scrum Team of what it means for work to be characterized as complete.

[3]The Sprint Goal is according to Schwaber and Sutherland (2017) "an objective set for the Sprint that can be met through the implementation of the Product Backlog".

- What functionality can be delivered in the upcoming Sprint?
- How will the work required to develop this functionality be executed?

It is the Development Team who works to estimate the functionality that will be delivered in the upcoming Sprint, while the Product Owner discusses the objective that the Sprint should achieve and the PBIs required to accomplish the Sprint Goal. However, it is only the Development Team that selects the number of items from the Product Backlog, as they are best suited to estimate what they can accomplish during the upcoming Sprint (Schwaber & Sutherland, 2017).

After defining the Sprint Goal and the PBIs for the next Sprint, the Development Team needs to figure out how this work will be executed. The Development Team often starts by designing the system, which at the end of the Sprint will be the product increment. Given that work may be of varying size or estimated effort, an essential aspect of the Sprint Planning involves decomposing the work into more manageable tasks, for example into units of one day or less. During this process, the Product Owner may assist the Development Team to clarify the selected PBIs and make trade-offs. Within the end of the Sprint Planning, the Development Team should be able to describe how they have planned to achieve the Sprint Goal and develop the product Increment as a self-organizing team (Schwaber & Sutherland, 2017).

Daily Scrum

According to Schwaber and Sutherland (2017), the Daily Scrum is a 15-minute time-boxed meeting where the entire Development Team gathers to plan work for the next 24 hours. This event is held daily and strives to optimize team collaboration and performance by inspecting the work done since the previous Daily Scrum and then estimating upcoming Sprint work (Schwaber & Sutherland, 2017).

It is the Development Team itself who decides the structure of the Daily Scrum. However, many Scrum implementations use at least the following three questions (Schwaber & Sutherland, 2017):

- What did I do yesterday that contributed towards achieving the Sprint Goal?
- What will I do today to contribute towards achieving the Sprint Goal?
- Do I see any impediments that may prevent the Development Team or me from meeting the Sprint Goal?

Daily Scrums serve various purposes. They seek to improve communication, identify impediments to development, monitor progress towards the Sprint Goal, eliminate the need for other meetings, foster quick decision-making, as well as to improve the overall knowledge of the Development Team. Members of the Development Team often meet after the Daily Scrum to further discuss topics brought up during the meeting or to adapt or replan the remaining work of the Sprint (Schwaber & Sutherland, 2017).

Sprint Review

The Sprint Review is an event held at the end of the Sprint to inspect the new product Increment and adapt the Product Backlog if required. This meeting is where the Scrum Team and stakeholders collaborate about what was achieved during the Sprint, as well as discuss the next actions that could be taken to optimize value. The length of a Sprint Review may vary, but it should not exceed four hours for one-month Sprints. As this is an informal meeting, as opposed to for instance a status meeting, the Sprint Review is primarily intended to elicit feedback and encourage collaboration (Schwaber & Sutherland, 2017).

The Sprint Review may include various elements, and some of them are (Schwaber & Sutherland, 2017):

- The Product Owner explains the work that was accomplished, including which PBIs that were "Done" and which were not.
- The Development Team demonstrates the new product Increment, as well as answers questions regarding it from the other attendees.
- The Development Team reviews their thoughts and experiences from the Sprint, including what went well, what problems arose, and how those were handled.
- Review of budget, timeline, potential capabilities, and the situation of the marketplace.

Sprint Retrospective

The Sprint Retrospective is a meeting where the Scrum Team inspects itself to discover potentials for improvement that can be implemented in the new, upcoming Sprint. The meeting takes place after the Sprint Review and before the Sprint Planning. Similar to the Sprint Planning and Sprint Review, the length of the Sprint Retrospective may vary. However, Schwaber and Sutherland (2017) suggests that the meeting should not exceed three hours for one-month Sprints.

The Sprint Retrospective serves various purposes (Schwaber & Sutherland, 2017):

- Examine the previous Sprint in relation to people, relationships, process, and tools.
- Uncover and order the things that went well and the potential improvements.
- Develop a plan for how these improvements should be implemented.

Product Backlog Grooming

Even though it is not an official Scrum meeting, Product Backlog Grooming (or Refinement) is something that many have discovered as valuable as it may lead to a more productive Sprint Planning (Cohn, 2015). Product Backlog Grooming is a meeting that is held near the end of one Sprint to make sure the Product Backlog is ready for the next Sprint. The meeting allows for the Development Team and the Product Owner to discuss the top items on the Product Backlog, and also allows the Development Team to ask questions that would typically arise during a Sprint Planning. Doing so enables the Product Owner to arrive at answers to questions that the person cannot answer immediately before the actual Sprint Planning (Cohn, 2015).

2.4.3 Scrum Artifacts

The Scrum framework also consists of some artifacts that illustrate work or value to provide transparency and possibilities for inspection and adaption (Schwaber & Sutherland, 2017).

Product Backlog

The Product Backlog is an ordered list of project requirements with associated time estimates for how long it will take to turn the requirements into working product functionality (Schwaber, 2000, p. 142). This list is dynamic, meaning that changes to it may occur during development to make the final product more suitable and valuable for the Product Owner. As the Product Backlog is never to be considered a complete list, it can only indicate the functionality and work defined up to that given point, and the final product may have several deviations compared to what was initially planned (Schwaber, 2000, p. 10).

Sprint Backlog

The Sprint Backlog is a selection of PBIs chosen for the given Sprint, as well as a plan for producing the Product Increment and achieving the Sprint Goal. The Sprint is an estimation by the Development Team of what functionality the upcoming Increment will contain, and the amount of work needed to achieve this. To monitor the progress of a Sprint, one can summarize the remaining work in the Sprint Backlog. This progress is tracked every Daily Scrum to estimate the likeliness of achieving the Sprint Goal (Schwaber & Sutherland, 2017).

The Sprint Backlog is characterized by being dynamic. This implies that the Development Team can modify the Sprint Backlog throughout the Sprint and that it emerges during the Sprint. It is only the Development Team that can change the Sprint Backlog during a Sprint. When additional work is needed, the Development Team adds it to the Sprint Backlog, and when work is performed or completed, the estimated remaining work is updated. Tasks that are deemed unnecessary are removed. The Sprint Backlog also includes one or more process improvements which were identified in the previous Sprint Retrospective meeting to foster continuous improvement (Schwaber & Sutherland, 2017).

2.5 Challenges of Agile Implementation

A systematic literature review conducted by López-Martínez, Juárez-Ramírez, Huertas, Jiménez, and Guerra-García (2016) uncovered 22 challenges related to the adoption of agile methodologies and Scrum. The study is based on 27 papers from 2012 to 2015 found on different scientific research libraries: IEEE Xplore, Science Direct, ACM DL, and Springer Link (López-Martínez et al., 2016). The findings of this systematic literature review are presented in table 2.1. The findings are organized into four different categories: organization, people, project, and process. Each challenge is also given an ID, which is used for reference in chapter 5.

ID	Key Challenges in the Agile Migration	Category	Number of Papers
C1	Organizational culture does not support agile ways of working	Organization	3
C2	Lack of capacity to change the organizational culture	Organization	2
C3	Organizational problems	Organization	2
C4	Lack of management support	Organization	1
C5	External pressure to use traditional practices	Organization	2
C6	Lack of collaboration and communication with the customer	People	2
C7	Lack of training of the Product Owner and the customer	People	1
C8	Team size	People	4
C9	Team unaligned	People	1
C10	Equipment capacity	People	1
C11	Rotating team members	People	2
C12	Lack of experience with agile methods	People	2
C13	Availability of trained personnel	People	2
C14	Lack of effective communication	People	3
C15	Lack of understanding of agile values	People	1
C16	Inadequate and dysfunctional training	People	1
C17	General resistance to change	People	2
C18	Lack of commitment to decisions	People	1
C19	Continued involvement with the client	People	1
C20	Project size	Project	3
C21	Agility degree	Process	1
C22	Anti-patterns	Process	1

Table 2.1: Overview of challenges uncovered in the strategic literature review, which category they belong to, as well as the number of papers that mention each challenge (Adapted from López-Martínez et al. (2016)).

According to López-Martínez et al. (2016), organizational culture is a crucial factor in order to achieve successful implementations of agile methodologies. Challenges such as organizational problems, lack of support from management, an organizational culture that does not support agile ways of working, absence of capacity to change the organizational culture, or pressure to use traditional practices from external parties may lead to suboptimal agile implementations. In order to achieve the most advantageous agile work process, the agile methodologies must be implemented within an agile culture that supports and embraces the agile values and principles (López-Martínez et al., 2016).

The systematic literature review also uncovered several challenges related to people. These challenges are: poor collaboration and communication with the customer, insufficient training of the Product Owner and the customer, the size of the teams, unaligned teams, equipment capacity[4], the rotation of team members, lack of experience with agile methods, availability of trained employees, lack of effective communication and understanding of agile values, insufficient and dysfunctional training, overall resistance to change, lack of commitment to decisions, and lack of continued involvement with the client (López-Martínez et al., 2016).

Although the majority of the studies included in this systematic literature review are focused on organizational aspects and people, a few also highlighted challenges with projects. A challenge mentioned within this category is the difficulty to scale in large projects (López-Martínez et al., 2016).

The systematic literature review also referenced two papers that discussed challenges of agile implementation related to the work process of agile teams. Lorber and Mish (2013) mention some challenges often found in early stages of agile adoptions, such as lack of deliveries of user stories[5], lack of confidence, as well as Sprint Planning meetings, Daily Scrums and Sprint Retrospectives lasting longer than intended with minimal value to the attendees. The study also highlights some more specific situations, for instance, that writing concrete user stories can be challenging due to the high degree of uncertainty (Lorber & Mish, 2013). In the other paper, Eloranta, Koskimies, Mikkonen, and Vuorinen (2013) uncover two types of deviations that represent unfavorable consequences for projects related to agile adoption, where the deviations are referred to as Anti-patterns. These two types of unfavorable deviations are (1) harmful deviations from recommended Scrum practices and (2) recommended Scrum practices that are for some reason unsuitable in a particular context. The study also states that both these types of deviations are undesirable since even though a Scrum deviation is well-motivated in a particular context, the deviation has harmful consequences in most cases. Examples of such deviations mentioned are that Sprint durations are too long, testing is done in the next Sprint, and poor documentation of specifications (Eloranta et al., 2013).

[4]The literature review calls this "Equipment capacity", but the article it cites only discusses team capacity.

[5]Mike Cohn, co-founder of Scrum Alliance, defines user stories as "short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system." (Cohn, n.d.)

Chapter 3

Method

This chapter presents and describes the methodological approaches used in the thesis. Also, the reasoning behind these approaches, as well as their strengths and weaknesses, are discussed. The chapter covers research design, literature review, case study, data collection, interviews, observations, and validity and reliability.

3.1 Research Design

A good research design is according to Easterby-Smith, Thorpe, and Jackson (2015, p. 8) fundamental for trying to achieve high-quality research, and they define research design as a written statement "which explains and justifies what data is to be gathered, how and where from." (Easterby-Smith et al., 2015, p. 68). The research design in this thesis is based on a qualitative, descriptive case study where primary data is gathered through semi-structured interviews and observations. Secondary data, gathered from a literature review, is used to compare the findings from the case study to what others have found before.

3.2 Literature Review

Due to the scope of this thesis and the limited time available to complete it, a separate literature review was not conducted. Instead, the findings from the systematic literature review conducted by López-Martínez et al. (2016) were used for comparison with the findings from the case study. This particular review was chosen because it is relatively new, and it selected papers from 2012 to 2015. The systematic literature review found 269 papers and evaluated 27 of them, and it focused on challenges with adopting agile. Additionally, some other single studies were used in the discussion to support the findings from the case study.

3.3 Case Study

A case study is a research method that according to Easterby-Smith et al. (2015, p. 89) "looks in depth at one, or a small number of, organizations, events or individuals, generally over time.". There can be either single case studies or multiple-case studies (Easterby-Smith et al., 2015, p. 89). While case studies are typically used when the research questions are of the "how" and "why" type, they can still be relevant for "what" type of questions (Yin, 2017). The method is preferred when it is not, or to a small degree, possible to control the behavior of participants in the study, and when the study focuses on a contemporary phenomenon (Yin, 2017).

Different types of case studies exist for different purposes, among them describing or exploring a case, or comparing cases (Baxter & Jack, 2008). While these types can cater to a range of different purposes, there are according to Yin (2017) still some traditional concerns regarding case study research: whether it is rigorous enough, confusion with non-research case studies, inability to generalize, advantages compared to other methods and level of effort. Level of effort refers to the tendency of researchers to answer too broad questions or to have too many objectives in their study, and as such Baxter and Jack (2008) specifies the need to keep the study in scope by placing boundaries for the case.

This thesis uses a single-case study that examines a company that has implemented agile in their system development department. It is a descriptive

type of case that relies on empirical data, and the case is bound by the department of the case company and the present time. The case study tries to find challenges that the department is experiencing, and then compares them to challenges found in similar situations in other studies. The company was chosen because it has been operating for a good number of years, and are at a stage where they have just completed a four-year journey of implementing and experimenting with agile software development.

3.4 Data Collection

The case study utilizes qualitative methods to gather information and data. Qualitative methods generate words instead of quantitative numbers as they try to understand some given phenomenon (McCusker & Gunaydin, 2015), and they, therefore, go well with case studies. When collecting qualitative data, the researchers are, according to Jacobsen (2015, p. 128), not actively guiding what kind of answers the respondents give, and the data is not categorized until they are collected. The different qualitative methods include, but are not limited to interviews, observations, documentation, and archives (Baxter & Jack, 2008). The methods used in this thesis is mainly interviews, with some supporting observations.

This thesis tries to uncover challenges that a system development department experiences; as such, the employees that work in the department are essential data sources. There was no available documentation related to challenges that the department experiences, so this method was not used. There was not enough time to conduct new interviews with the respondents, so some clarification questions were sent and answered by email by the Application Development Manager at the case company.

3.4.1 Interviews

Interviews are, according to Yin (2017), one of the most important sources of data for case studies. "Qualitative interviews are directed conversations evolving around questions and answers about a certain topic" (Easterby-Smith et al., 2015, p. 133), and what separates them from normal conversations is that they follow a series of questions to explore an event or topic

in-depth (Easterby-Smith et al., 2015, pp. 133-134). The advantage of using qualitative interviews is that they let the researchers gain an understanding of the respondents' perspectives and viewpoints. Interviews also make it possible to discover data that is neither documented, archived, nor possible to observe (Easterby-Smith et al., 2015, p. 135). Drawbacks of qualitative interviews are their complexity in terms of time commitment and fitness to purpose (Easterby-Smith et al., 2015, p. 139), and that they can suffer from subjective problems such as bias, poor recall, lies and poor articulation (Yin, 2017).

Yin (2017) talks about three different types of case study interviews: prolonged, shorter, and survey interviews. The prolonged interviews last two or more hours in single or multiple sittings and are open-ended. The shorter interviews typically last around an hour and can be open-ended while still following an interview protocol more closely. Survey interviews follow a structured questionnaire where the result may be quantitative data (Yin, 2017). Easterby-Smith et al. (2015, p. 139) also mentions three different types of interviews based on how structured they are. Highly structured interviews have detailed and structured interview protocols with some narrow answer selections, semi-structured interviews have an interview protocol that contains topics to talk about, and unstructured interviews only has some questions to stimulate conversations (Easterby-Smith et al., 2015, p. 139).

Interviews were conducted, individually, in this case study to gather as much data as possible about the challenges the system development department experiences. This was accomplished by interviewing employees with different roles in the department. One interview was conducted with each respondent, and the roles interviewed were (see section 4.4 for an explanation of their responsibilities):

- Application Development Manager
- Technical Product Owner
- Software Developer x2
- Test Team Manager
- Technical Test Manager

All the roles in the department, except for tester, were interviewed. The reason for why a tester was not interviewed was because the Technical Test Manager also works as a tester. It was therefore assumed that the Technical Test Manager had the same knowledge and experiences as the other testers. Furthermore, only software developers from one of the development teams of the software development department were interviewed due to time limitation and available employees. The structure of the Software Development Department is explained in section 4.2. Corroborating the interview data with other sources neglects some of the drawbacks of interviews (Yin, 2017). This was, however, only partly done with observations as it is not so relevant to corroborate when the interviewees' personal views are of interest (Yin, 2017). Most of the challenges uncovered were also identified by two or more of the interviewees.

The interviews conducted were semi-structured. The reason behind this was that there were some questions given that had to be answered, but the interviewees were free to talk open-ended. The interview protocol was mostly equal for each respondent, but some questions were adjusted or not asked based on the respondent's role in the department. Certain formulations and follow-up questions were added based on answers given to the other questions. An interview protocol with all questions can be seen in appendix A. Depending on how much the respondents answered and what they said, follow-up and new questions were asked to clarify or to gather more information. The interviews can be classified as short, as they lasted between 15 and 85 minutes. All the interviews were audio recorded and later transcribed, resulting in just over 30 000 words in total.

3.4.2 Observations

Observations are according to Jacobsen (2015, p. 165) about registering what people do, and not just what they say they do. He also says that observations can avoid some of the subjective problems interviews may suffer from, but they give no insight into what people think or mean (Jacobsen, 2015, p. 165). Easterby-Smith et al. (2015, p. 162) mentions four types of stances observers can take: complete observer, observer-as-participant, participant-as-observer, and complete participant, and they differ in how involved the observer is in what they observe. All observations should be recorded in some way, ranging from video recordings to notes (Easterby-

Smith et al., 2015, p. 162). Observations can be used for evidence in case studies, and to corroborate findings from other sources (Yin, 2017).

This case study employed observer-as-participant observations to corroborate on findings from the interviews. Two different Scrum meetings, Sprint Planning and Daily Scrum, were observed, and notes were used to document the findings. The participants of the meetings were aware of the case study and its purpose, and the only questions asked from the observers were for clarification purposes during one of the meetings.

3.5 Validity and Reliability

There is no doubt that quality and rigor play an integral role in all research, qualitative research included, but there are variations in how the quality should be judged (Ali & Yusof, 2011). The stances on quality range from (Ali & Yusof, 2011): "There is only one way to judge the quality of qualitative studies which is the same for any type of scientific inquiry: the criteria of reliability, internal and external validity and objectivity." to "There is no way to judge the quality of qualitative studies.". The three prominent case study methodologists Robert Yin, Sharan Merriam, and Robert Stake all differ in their views on validity, and their differences are based on their viewpoints of research (Yazan, 2015).

Validity has often been split into internal and external validity by primarily quantitative researches, where internal refers to "the degree to which the results can be attributed to treatment" and external to "the generalizability of the results" (Ali & Yusof, 2011). Yin (2017) does, however, state that internal validity is not relevant for descriptive case studies. Another view of validity is according to Easterby-Smith et al. (2015, p. 103) concerned with answering the question: "Have a sufficient number of perspectives been included?", while generalization is concerned with answering "Is the sample sufficiently diverse to allow inferences to other contexts?".

For this case study, it might be possible to generalize the challenges that were only observed in the case company. However, there might also be too much specific context in the case to make such claims. The underlying reasons and consequences might, however, be more suitable for generalization, as they are not bound as much to the context of the case company as they

are to the challenges themselves. Regarding the number of perspectives, there has been conducted interviews with almost all roles in the system development department to receive the widest array of challenges facing the department. Interviews have, however, been the primary data gathering method, with some observations. More challenges could potentially have been uncovered by using other data gathering methods such as documentation. While not part of the thesis, interviewing the business side of the case company could have uncovered challenges in the department that the employees there have not noticed.

There is less discussion on the meaning of reliability, as most definitions are quite similar: "a matter of degree of consistency of observed objects agreed upon by one observer on different occasions or by different observers" (Hsieh, 2004), "the extent to which the findings can be replicated" (Ali & Yusof, 2011), "demonstrating that the operations of a study—such as its data collection procedures—can be repeated, with the same results" (Yin, 2017), and "Will similar observations be reached by other observers?" (Easterby-Smith et al., 2015, p. 103). It is clear that reliability is focused on whether the results of a case study can be replicated or not, and this can be influenced by data gathering methods, the trustworthiness of the gathered data, current contexts, the interpretation of data and documentation of research method.

The research method in this thesis is described, and the interview protocol is added in appendix A. The protocol does not contain all follow-up questions, and some context and answers might, therefore, be hard to replicate. Replicating interviews is also hard, as conversations can be affected by mood, voice, body language, and trust level. Using other data gathering methods such as anonymous surveys or documentation might also lead to uncovering other challenges, and using more method triangulation would lead to more trustworthy data. While all the data used in the discussion chapter is based on information from the transcribed interviews, there has been interpretation involved when structuring and merging challenges. As two researchers have interpreted the data, it is reasonable to expect that the reliability following the interpretation is high. It is, however not possible to know whether the interviewees told the truth or not, but there were no conflicting statements made regarding their challenges, and there is no apparent reason for why they should lie.

Chapter 4

Case Study

This chapter presents the company used in the case study (hereafter Case Company) and serves as an explanation of all relevant information used later on in this thesis. First, some general information about the Case Company is presented. Secondly, the structure of the system development department is explained. Thirdly, their agile transformation is described. Fourthly, a list of roles and their responsibilities in the system development department is given. Then lastly, the work methods and approaches used by the Case Company are described. The information in this chapter is based on data gathered through interviews and conversations with the employees at the Case Company.

4.1 Case Company

The Case Company operates in the Norwegian financial sector, has over 1500 employees, and in 2018, its profit before tax was over 3000 million NOK. The company is a subsidiary of a multinational organization but operates rather individually from their parent company. They are free to choose their internal organizational structure and work methods, however, they must follow certain security and financial policies given by the parent company. The company provides a wide range of services and products to both end consumers and other businesses.

4.2 Department Structure

The Case Company has a system development department that consists of three subdivisions, one internal and two external. The external subdivisions consist of multiple consultant teams containing developers, testers, and technical product owners. They are, within the framework given by the Case Company regarding methodology and programs, self-organized. They have their own experiences with agile approaches, and may, therefore, follow their variation of agile. The two external subdivisions are excluded from this thesis (as previously explained in section 1.3). For the rest of this thesis, the "System Development Department" will, unless specified, only refer to the internal subdivision of the system development department of the Case Company. The System Development Department consists of two development teams, .NET and database, and one team consisting of testers and a Technical Test Manager. The employees in the department are a mix of full-time employees and consultants. The System Development Department has responsibility for a critical back-end system that is used by a large part of the Case Company's services and products. The structure of the System Development Department is shown in figure 4.1, and the roles that were interviewed are shown in white boxes in the same figure.

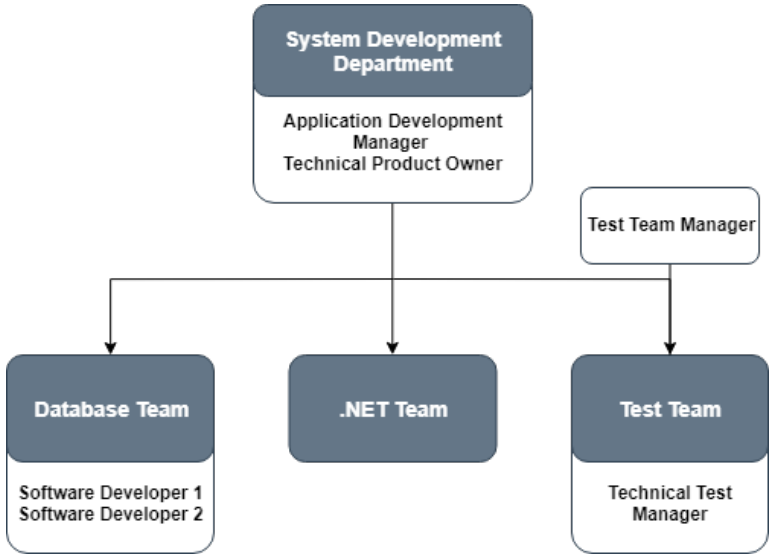


Figure 4.1: Structure of the System Development Department, and overview of the interviewed employees.

4.3 Agile Transformation

Due to unsatisfying results, long development times, poorly described tasks, discontent from the company about not receiving what they asked for and burndown charts[1] that did not show the desired results, the System Development Department started to experiment with new development methods a few years ago. Most of the methods were taken from agile approaches, and the most notable elements were shorter development periods and breaking down tasks into smaller, more manageable tasks. As such, they were slowly moving to an agile approach. However, they still had challenges completing the tasks that they were committed to in each Sprint, and their burndown charts were still not showing the desired results.

About four years ago, the department hired an external, agile coach to evaluate their development approach. The coach conducted interviews, acted as a kind of Scrum Master, and gave feedback on how they could change their development methods. This lead to more down-scoping of tasks and a change in mindset, resulting in improved development times and improved burndown charts. Since then, the System Development Department has considered themselves as agile. They now claim they follow a version of the agile development framework Scrum, even though they do not do it rigorously. The coach also split the single Development Team into two separate development teams. This was due to the developers working on two separate aspects of the system that the department is responsible for managing.

The System Development Department started to hire Scrum Masters after the agile coach left. They have so far had two different Scrum Masters. As the department was and still is, free to choose how they work, they give much power to their Scrum Masters when it comes to what they are allowed to do and change. This freedom lets the Scrum Masters experiment with what works best, and as such, the department optimizes how they work. The first Scrum Master followed Scrum somewhat rigorously, while the second Scrum Master only followed what he deemed to be essential and beneficial.

While the System Development Department is working agile with their customized version of Scrum, the rest of the company still operates more tra-

[1] A visual tool that shows completed work per day compared to the projected completion rate (Scrum Institute, n.d.).

ditionally. The top management of a foreign branch office has taken agile to heart, and this has affected the top management in Norway who are now trying to implement agile in a larger part of the Case Company. Currently, the company has some non-development teams that are trying to work agile, and the goal is to transform their company into being business agile[2].

4.4 Roles and Responsibilities

The System Development Department consists of various roles that each have its own set of responsibilities. The following paragraphs will describe the different roles in the System Development Department, as well as other relevant positions that are connected to the department.

Applications Development Manager

The Applications Development Manager is the head of the System Development Department and is responsible for all the system and application development within the Norwegian business unit of the company. This person works closely with the various Technical Product Owners to make sure the different development teams deliver functioning software in accordance with project plans. The Applications Development Manager is also responsible for bringing in new employees and consultants when needed, as well as making sure this does not exceed the budgets of the different projects the department works on.

Technical Product Owners

The Technical Product Owners are persons that function as the connecting link between the development teams and business. Each Development Team has its own Technical Product Owner who is responsible for translating the orders and requests the team receives from the Product Owner from business into PBIs that better describe what is to be developed on a more technical level. The Technical Product Owners are often senior developers with a

[2]The aim of business agility is to maintain a competitive advantage in uncertain times by responding quickly and adapting to the environment (Mathiassen & Pries-Heje, 2006).

type of architectural role in the sense that they possess knowledge on how the entire system landscape of the Case Company is built up and thereby know how their system behaves and functions in relation to other systems. The Technical Product Owner in the System Development Department is also responsible for planning each Sprint, as well as ordering the PBIs they receive from different projects in accordance with the prioritization set by the project management office.

Product Owner

Since most projects the System Development Department participates in come from other departments in the Case Company, these departments become the project owners of the various projects. The Product Owners are the responsible person for a project that the department can communicate with, and they are also the people responsible for bringing PBIs to the Technical Product Owner.

Software Developers and Testers

There are primarily three types of tasks that the software developers and testers are responsible for doing. Firstly they develop new solutions and functionality for different projects. Secondly, they maintain their system and fix problems and bugs that affect it. Thirdly the testers check all new features and changes done to the system. Moreover, the testers also do testing and quality assurance on other projects not related to the two development teams. The size and scope of the projects the teams participate in vary, and the tasks can range from developing new features to making small configuration changes to comprehensive end-to-end testing. There are five developers in the database team, four in the .NET team and four testers in the test team.

Test Team Manager

The Test Team Manager is the head of the testing department. All Technical Test Managers report to the Test Team Manager, who is responsible for coordinating the test environments.

Technical Test Managers

There are four Technical Test Managers in the Case Company. One of them works in the System Development Department, while the three other Technical Test Managers are responsible for testing within their systems. The Technical Test Manager for the System Development Department is responsible for distributing the various PBIs for a given Sprint among the testers, as well as coordinating which project each tester is working on during the Sprint. As the group of testers are few in total and mostly consists of consultants, they have to be shared between the various projects that are happening in parallel.

Scrum Master

The System Development Department also has a dedicated person who serves the role as Scrum Master. This person is responsible for facilitating the different Scrum meetings, removing impediments that may have an adverse effect on the development teams' progress and help the teams to work according to Scrum theory.

The System Development Department is at the moment without a dedicated Scrum Master since the contract of the previous one expired. Not having a dedicated Scrum Master has led to the Technical Product Owner taking over these responsibilities in addition to those he already has. This situation is, however, only temporary as the department is already looking for a replacement.

4.5 Work Methods

As mentioned in section 4.3, the System Development Department has over time adopted their own version of Scrum. The development teams and the Scrum Master are together given the freedom to decide how strictly they follow the framework. The development teams optimally try to stick to Sprint intervals of two weeks. However, the teams sometimes have to extend the Sprint by an extra week during periods with a high workload. The reason for this is that the development teams, in addition to developing new

product functionality on its platform, also is responsible for fixing potential bugs discovered in the production environment. Before each Sprint, a Sprint Planning meeting is held where everything that is to be done in the upcoming Sprint is planned in detail. The Technical Product Owner invites members of the team to define and describe the tasks within each PBI, so they are easily understandable and clear. The Technical Product Owner has prior to the Sprint Planning meeting had another meeting with the Product Owner from business to ensure the focus of the next Sprint is aligned with the prioritization of the Product Owner and the overall strategy of the organization.

During a Sprint, Daily Scrums are organized. They are, however, not always held daily. After the last Scrum Master finished their contract, the regularity of the Daily Scrums reduced over time, mostly because only some of the team members took the initiative to organize these meetings. The same also applies to the Sprint Reviews and Sprint Retrospectives. The Sprint Reviews are sometimes held if the Product Owner wants it, and the Sprint Retrospectives are just deprioritized and not held. As such, the department is not rigorously following Scrum, but they do instead use the elements they feel give them the most benefit.

As the testers in the System Development Department are not part of the two development teams, they are doing PBIs and tasks from both teams' Sprint Backlogs.

Chapter 5

Discussion

This chapter analyses and discusses the findings from the interviews conducted in the case study. The findings of this study have been combined with the discussion to provide a cohesive presentation of them, as it might be difficult to make sense of the findings alone without accompanying interpretation. The following section gives an overview of the findings, while the subsequent section presents the challenges uncovered at the Case Company, as well as discusses them in relation to existing theory, their causes, and consequences and how to possibly mitigate them. In the end, a section summarizes the discussion.

5.1 Findings

After conducting and transcribing the interviews, all the uncovered challenges were listed individually. Similar challenges were combined, and those that were direct consequences or causes to others were merged. This process reduced the number of challenges found from 37 to nine. As mentioned in section 1.3, challenges not related to agile or its implementation were discarded. Figure 5.1 shows the merged list of challenges, as well as how many of the interviewees that mentioned them.

Challenge \ *Person*	Technical Product Owner	Software Developer 1	Test manager	App Development Manager	Software Developer 2	Test Team Manager
Sprint Workload		x		x		x
Testing in the Next Sprint				x		x
PBI Descriptions	x	x	x	x	x	x
Business Agility	x		x	x		
Documentation	x				x	
PBI Grooming	x	x			x	x
Team Improvement		x			x	
Release Process				x		
Sprint Review					x	

Table 5.1: Overview of challenges uncovered during the interviews, and which participant that mentioned them.

5.2 Challenges

This section goes through the challenges that were uncovered and explains how they are challenging. Every challenge is described, their potential causes and consequences are discussed, as well as possible solutions are presented. Related challenges found in the systematic literature review are labeled with their ID, as defined in table 2.1.

5.2.1 Sprint Workload

A challenge mentioned by three of the interviewees is that the two development teams have too much work to do in each Sprint. The challenge does not happen every Sprint, but frequently enough to be challenging for the department.

A common reason for why there is too much to do in a Sprint is that development teams are imprecise when estimating the length of PBIs and tasks (Popli & Chauhan, 2013). The agile coach that the Case Company had previously taught the development teams to estimate by complexity instead of time. By learning from previous mistakes, this has become a skill that the teams are good at, and as such, the reason for why the teams have too much work in their Sprints lies elsewhere. Another possible reason, which is also prevalent in existing theory (C10), might be that the team

capacity is lower than what is needed.

One cause is that each Sprint allocates a set amount of time to maintenance work. The maintenance work that shows up has to be completed immediately. Seeing as it is impossible to estimate how much maintenance work will appear in a Sprint, there is a possibility that there will be more than estimated. This results in other tasks having to be moved to the next Sprint, which is more thoroughly discussed in section 5.2.2.

Having too high workload can lead to different consequences such as certain PBIs being moved to the next Sprint, extended Sprints, and unsatisfactory burndown charts. All these consequences can lead to further potential challenges like delays in projects, the next Sprint being interrupted by bugs discovered too late, poor quality and exceeding budgets (Popli & Chauhan, 2013). A large consequence that can happen is that testing is moved to the next Sprint, see section 5.2.2 for further details. Kniberg (2015) mentions that too large PBIs ends up being partially complete, which does not produce value for the company and leads to more administration.

To solve the challenge of having too high workload in a Sprint, the development teams sometimes extend their Sprint duration from two to three weeks. While this is a reactive measure that does not fix the causes of the challenges, it makes sure that the Sprint is completed. It does, however, have the consequences of delaying the next Sprints, which impacts the projects that have development work planned for those Sprints. A preventive solution to the challenge is to break down PBIs even more so that they fit inside the scope of a single Sprint. Another possible solution is to adjust the allocated amount of time given to maintenance tasks. This solution may, however, have the opposite effect on the challenge, meaning the teams end up with too little work to do in each Sprint. Analyzing when and why there is much maintenance work might lead to better allocations of maintenance work for each Sprint. Estimation of maintenance work is also something not accounted for in the Scrum framework, and can, therefore, be challenging to manage.

5.2.2 Testing in the Next Sprint

A challenge mentioned by the Test Team Manager is that testing is not always done in the same Sprint as development. This situation was also explained by the Application Development Manager, but was, however, not labeled as a problem. According to Scrum theory, on the other hand, every Sprint should produce a "Done", usable, and potentially releasable Product Increment (Schwaber & Sutherland, 2017), which implies that testing has to be done within the same Sprint as development. This practice is also mentioned by West, Gilpin, Grant, and Anderson (2011), who states that testing is often moved outside the given Sprint and also often to another separate team, which contradicts with agile principles.

One reason for why testing and development is not done in the same Sprint is that there might not be enough time in the Sprint to do both. Sometimes this is known in advance, and in such cases, there are often separate PBIs for development and testing. The PBI for testing is then placed in the next Sprint. Other times this is not known in advance as the time constraint is discovered at the end of a Sprint, resulting in the test task having to be moved to the next Sprint. Some other causes of this problem, like an unpredictable amount of maintenance work, has already been covered in section 5.2.1.

Not having development and testing in the same Sprint means that when testing starts in the following Sprint, the developers might already be working on new PBIs. So if a bug or unwanted behavior is discovered during testing, the developers might have to stop what they are doing so they can fix the issue. Having to fix issues that appear disrupts what they are currently working on, which in turn may lead to the current task being delayed. Delays in the development during a Sprint can result in not enough time to do testing, meaning testing again is moved to the next Sprint. These delays can form a vicious circle which might be hard to break out from, and it might be a sign that the department does not fully understand the agile values (C15). Having testing and development in the same PBI, but different Sprints also negatively affects the burndown chart, as it only counts completed PBIs in a Sprint.

Another similar consequence mentioned by Eloranta et al. (2013) of having testing in the next Sprint, which is referenced in the systematic literature

review (C22) in section 2.5, is that new functionality might already be written on top of old code which is still in the testing phase. Changing untested code may again result in additional time and resources being wasted to fix potential bugs in the old code. In the worst case scenario, the new code might be left useless and obsolete. A third consequence of splitting development and testing from the same PBI into different Sprints is that it can lead to a varying definition of "Done", which may cause confusion within the development teams.

A way to avoid having to split development and testing due to a PBI being too large for a single Sprint is to break down the PBI into smaller PBIs. By breaking down the PBI into smaller PBIs, it is easier to do both development and testing in the same Sprint, and then do the same to the other PBI in the next Sprint. A preventive measure for the cases where it is not apparent until the end of the Sprint that there will not be enough time for testing, is to be better at estimating the complexity of the PBIs. Estimating better can either be done by adding more complexity if there is uncertainty about a PBI, or by continuously learning from previous Sprints and PBIs so that the accuracy of estimates increases. A reactive measure for the same case is to extend the duration of the Sprint. Extending the Sprint can, however, lead to other problems such as delays, re-prioritization of projects and PBIs and the need for re-planning. Even though this might lead to negative consequences, extending the Sprint duration is sometimes done by the Case Company.

5.2.3 PBI Descriptions

The largest challenge that was emphasized during all the different interviews was how the PBIs from the Product Owners at the business side often are poorly described and detailed. The Applications Development Manager mentioned a challenge that they sometimes see is that business is experiencing that they do not get software, meanwhile, the System Development Department might say that things are too poorly described and are therefore having challenges committing to the given tasks. Software Developer 2 stated that there are some Product Owners from business that deliver orders that often lack adequate detailing. Software Developer 2 also said that the Product Owners from business only vaguely describe what they want, and leave out additional and often useful information, such as why it

is needed and how it should function. The Technical Product Owner also mentioned that the PBIs from business are often poorly defined and lack detail. PBIs without a sufficient level of detailing make it more difficult for the Technical Product Owner to break them down into more manageable PBIs due to the high level of ambiguity.

There might be several underlying reasons for why the PBIs from business are poorly described and detailed. One of the reasons might be that the organization does not have a company-wide standard on how PBIs should be described and formulated. This opens up for a lot of diversity from the various Product Owners when it comes to describing their feature requests. Lack of training, and trained personnel are repeating challenges in the literature (C7, C13, C16), and may also be a reason for why the Case Company has poorly described PBIs.

Another reason might be that it exists a lot of uncertainty when it comes to what is to be the final product, and where the solution to the business requirements can be difficult to define. A high degree of uncertainty makes it difficult to thoroughly describe what is to be developed, thus making the PBIs less concrete. This challenge is also mentioned by Lorber and Mish (2013), and is also included in the systematic literature review. They say the inherent uncertainty makes it hard to define concrete user stories, opening up for ambiguity of what is to be done and the acceptance criteria (Lorber & Mish, 2013). Lack of understanding of agile values (C15) might be an additional underlying reason.

Other reasons might be that the employees in business lack the required level of technical knowledge regarding the Case Company's systems to properly describe what the current problem is and what needs to be done in order to fix it. It might be that the System Development Department does not make strict enough demands of what is an accepted level of detailing regarding orders from business. It might also be that the project descriptions are being poorly described due to suboptimal communication and collaboration between business and the System Development Department (C6).

Poorly defined and detailed PBIs may lead to several unfortunate consequences. The Technical Product Owner mentioned that it often results in prolonged development times as additional meetings and questions back and forth between the development teams and business is required for them to clarify ambiguities and have a similar idea of what the finished product will

be. This may also lead to finished products that have large deviations compared to what business initially intended, which again can result in budget and time overruns.

A direct consequence of this challenge, mentioned by the Test Team Manager, is that testers do not exactly know what to test when they receive a task. While they can test what the developers made, they are not necessarily sure if that is what the Product Owner wants. It also makes it harder to see how other systems are affected by the changes. This can on its own lead to critical system bugs which can cause damage to the profits and reputation of the Case Company. This consequence may also be caused by inadequate PBI grooming, which is covered in section 5.2.6.

To avoid or minimize the risk of dealing with poorly defined PBIs, the organization could introduce a company-wide standard for how software feature requests or PBIs should be described and detailed. A company-wide standard could, for instance, be user stories or similar methods and tools that better lay out what the requested software should do and how this could be validated. Having a standard will also define a minimum of what is to be considered as an acceptable level of detail, thereby making better use of the software developers' time. However, this will also require sufficient training and education of Product Owners, so that they can write more extensive software development requests. It is also important that business is organized in such a way that they have the right type of people with the required knowledge and expertise to write good enough specifications.

Another important aspect mentioned by Cockburn (2002), one of the founders of the Manifesto for Agile Software Development, is the advantages of communicating face-to-face. As this type of communication is the most efficient, the development teams and the customer are encouraged to have face-to-face discussions when defining the specifications. Communicating requirements using User Stories only makes the communication one-directional, which can be inefficient and make it hard for the development teams to ask questions back (Cockburn, 2002).

5.2.4 Business Agility

Another challenge that was uncovered during the interviews was how the rest of the Case Company still operates in accordance with traditional project management approaches, and how this affects the System Development Department. The Technical Product Owner mentioned that there still exist many people within the organization that rather prefers the Waterfall approach to project management and software development. The Technical Product Owner also stated that one could not fully take advantage of the various benefits of working agile if the organization as a whole does not adopt the agile values and principles. The Applications Development Manager, as well as the Technical Product Owner, also mentioned that the organization is now in an ongoing process of making business think and work more according to the agile principles and values. This challenge of still operating after traditional approaches may be an indication that business lacks experience with agile methods (C12), and that they might not yet have an extensive understanding of agile values (C15).

A potential reason for this challenge may be related to the organizational culture of the Case Company. Struggling to think agile is often evidence that there still exist old culture within the organization. Meaning that some people within the organization are skeptical of changes (C17) that may affect how they execute their daily work. Organizational culture is also emphasized as a challenge (C1) in a study conducted by VersionOne (2012) on agile methods. The study, which is mentioned in a paper in the systematic literature review, gathered data from over 4000 survey participants. Of the respondents with failed agile projects, 12 % said that the main reason was due to company philosophy or culture being at odds with core agile values (VersionOne, 2012). Furthermore, the study also lists barriers to further adoption of agile practices, where 52 % of the total respondents stated that the "ability to change organizational culture" was a significant barrier. Introducing changes that will affect a company's organizational culture is a comprehensive process that requires that the majority of the organization contributes in order for the changes to take effect.

As the Technical Product Owner stated during the interviews, a potential consequence of not adopting the agile values throughout the organization, is that one cannot fully take advantage of the benefits that agile may offer. Not doing so may result in some departments within the organization becoming

bottlenecks to the progress of various projects, for instance, by prolonging the time before each new Increment is available to the customer or end consumer.

A possible way to avoid or mitigate this challenge is to change the employees' daily work routines to promote agile principles and values better. However, making business agile is not something that is done overnight. It requires continuous work and a constant focus on agile principles and values. A transformation like this also requires the participation of the majority of people within the organization. It is, therefore, crucial that this information is distributed across the entire organization. The information should emphasize the potential benefits of working according to the agile philosophy. It might also be relevant for the organization to introduce different types of incentives that promote working in accordance with agile principles and values.

5.2.5 Documentation

Another challenge that was uncovered during the interviews was how documentation often was deprioritized. The Technical Product Owner mentioned that writing documentation was often left out during periods with high workload and several projects going on in parallel. Software Developer 2 stated that the documentation that already exists is often outdated and, therefore, serves little use.

The root cause of this challenge may be traced back to the Manifesto for Agile Software Development and a lack of understanding for one of its values (C15). One of the four values states that working software is valued more than comprehensive documentation (Beck et al., 2001). The Manifesto does, however, not say that writing documentation is unimportant, only that functioning software should be prioritized higher. This formulation opens up for interpretation, and the amount of documentation may, therefore, vary significantly between various organizations.

Agile's view on documentation has, therefore, been a concern for organizations that consider adopting agile methods. This is evident in the study conducted by VersionOne (2012) (also mentioned in section 5.2.4), where 26 % of the respondents said that "lack of documentation" was a big concern when they were considering deploying agile. Another study by Prause and

Durdik (2012), which interviewed 37 software engineering experts from industry and academia about architectural design and documentation in agile development, concludes that there is indeed a problem with both.

There might be several reasons why documentation is being deprioritized and outdated. One of these reasons might be connected to the pressure some businesses put on their system development departments when it comes to delivering functioning software. Management wants to have functioning software as soon as possible in order to sell the product to its customers and might, therefore, push the development teams to deliver it faster than they usually would. As writing documentation only contributes a tiny part towards the progress of a project, the result may be that it becomes deprioritized.

Another reason may be that the time it takes to write documentation is not taken into account, or at best strongly underestimated, when it comes to calculating the workload for a Sprint. Two final reasons may be related to the culture of the organization (C1). The process of writing and updating documentation may have been something that always had low prioritization within the organization. And the people responsible for documentation might not see the value in it, or might not care about it.

Several of the findings discussed above are also prevalent in existing theory on the topic. For instance, Prause and Durdik (2012) uncovered in their study that the most likely reasons for challenges with documentation were due to the fact that developers might not care about it, do not know how to do it properly, lack time, do not explicitly consider documentation and design, have limited personal benefit, and miss defined quality goals.

One of the most severe consequences of inadequate and outdated documentation is the amount of extra work required when changes are needed. One example is trying to change old code that no one longer knows or understands. Another example is when an employee, who possesses expert knowledge about a given system, leaves the company. This lack of knowledge may also lead to lack of control and money wasted when trying to figure out how the system functions.

Deprioritized and outdated documentation can be avoided by setting aside enough time for documentation during the development process. Going through all the relevant documentation regularly can also help avoid out-

dated documentation. It is also essential to have good communication between the System Development Department and business to ensure that documentation is not deprioritized at the expense of faster delivery of functional software.

5.2.6 PBI Grooming

Mentioned by all but the Application Development Manager and the Technical Test Manager is the challenge of inadequately groomed PBIs. As both the developers and the testers use PBIs, this challenge affects both of the development teams and the testers assigned to these PBIs. Currently, the development teams are doing minimal grooming during their Sprint Planning meetings. Cohn (2015), co-founder of the Scrum Alliance, says a good rule of thumb is to spend between 5 to 10 % of an entire Sprint effort on grooming.

One reason behind the inadequately groomed PBIs is the lack of PBI grooming meetings. While these meetings could be held before a Sprint ends, they very rarely are. None of the development teams want to sit in long meetings, and as such grooming of PBIs is not something that is prioritized. A temporary lack of a dedicated Scrum Master might also be a reason for why the grooming meetings are not held (C13). Cohn (2015) mentions that long meetings with unprepared and unneeded participants reduce the value of grooming meetings. He also mentions the importance of having the Product Owner participate in the grooming meetings (Cohn, 2015), because there is a need for collaboration and communication with the customer (C6). What often happens in the System Development Department is that the PBI is only "groomed" until one team member understands what he or she is most likely supposed to do. Some team members may not see the point of grooming PBIs more than necessary, as the developer that knows the type of task best does the PBIs. That developer does not need much detailing on their PBI. Not adequately grooming PBIs may also contradict one of the agile values, which states that "Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely." (see section 2.3). Since maybe only one of the developers understands the PBI, this may lead to variable pace. As previously mentioned, this may be due to a lack of understanding of agile values (C15).

Increased specialization is a negative consequence of inadequately groomed PBIs. The PBIs are often only groomed until one of the developers in the team knows what is to be done. If the single person who understands the PBI is not available, then no one else can do it as they do not know what they are supposed to do. Increased specialization leads to the department becoming person dependent. Developers are not the only ones affected by the inadequately groomed PBIs, as testers will also have a hard time knowing what to test based on the PBIs. The challenge the testers face due to poor PBI grooming, and its consequences, is discussed in section 5.2.3.

There are at least two possible solutions for inadequately groomed PBIs. One is to ensure that the PBIs are better detailed when they arrive, or do more grooming of them either during Sprint Planning meeting or separate PBI grooming meetings. While the second solution can be implemented by the development teams, the first solution requires better requirements from the rest of the company, which may not be easy to accomplish (as discussed in section 5.2.3). There have been earlier attempts at improving the quality of the PBI grooming. However, the quality improvements were only temporary.

5.2.7 Team Improvement

Both of the software developers said the teams are not working enough with improvement. The challenge is twofold as earlier improvement practices have died out, and new attempts die out after a while. Earlier the teams did pair programming[1], which contributed to spread knowledge and the creation of better solutions. This practice is, however, no longer given as much time and priority. Other attempts at improvement such as better grooming of PBIs only work for a short while, before the teams fall back to earlier habits. According to Software Developer 2, the teams are not conducting Sprint Retrospectives anymore, even though they are an integral part of Scrum that is meant for discussing and implementing improvements (Schwaber & Sutherland, 2017). Not conducting these meetings is a deviation from Scrum (C22) and may suggest a lack of understanding of agile values (C15).

[1]Pair programming is a technique within agile software development where two developers share a single workstation. One of the programmers sits behind the keyboard and does the actual typing, while the other is more focused on the overall direction. The roles are swapped between the two when required (Agile Alliance, n.d.-a)

While the interviewees did not point to any specific reason for why they fail to improve, it was, however, mentioned that they were better before. The fact that they have become worse at improving may imply that a lack off a dedicated Scrum Master (C13) is a reason for why the team improvement is stagnant. A lack of dedicated Scrum Master may also be linked to a lack of Sprint Retrospectives, which in itself is a challenge. Sprint Retrospectives let the team members reflect on what they do well and bad, what they should improve, as well as how to improve on it. Dikert, Paasivaara, and Lassenius (2016) found Sprint Retrospectives to be used by many organizations for improvement in their literature review. Other possible reasons might be that the teams are complacent and not believe that they need improvement, or that they do not feel they have the time for improvement practices.

Specialization may be a consequence of having poor team improvement. This is because a part of team improvement is to spread knowledge within the Scrum Team. Specialization was found by Moe and Dingsøyr (2008) to have a negative impact on team orientation. If there is only one person that knows how a particular system, language, or piece of code works, there will be challenges if the person quits or is absent. A lack of improvement is also unfortunate for the Case Company as it will not achieve the full potential of its System Development Department. If improvement practices keep dying out after a while, the result might be that the employees stop trying to improve entirely, as they know that whatever they are trying will be abandoned in the future.

Similar to other challenges, one of the possible solutions to poor team improvement is to hire a dedicated Scrum Master. That will in itself not fix the challenge, as the Scrum Master must be able and willing to make them do and continue doing improvement practices. However, having a Scrum Master increases the chance of Sprint Retrospectives being held. Teaching the employees why they must improve and how different practices affect their improvement can also help to keep improvement initiatives alive. Forcing the employees to do improvement practices may also work, but it is important that the improvement is greater than the possible negative consequences that can follow when employees are forced to do something they rather not do. Setting aside more time for improvement might be a possible solution, but it requires sacrificing some amount of work between each Sprint. The improvement can, however, make this worth the time as it may result in more efficient work and better handling of situations where persons are absent or quit.

5.2.8 Release Processes

In the early stages of the organization's agile adoption, the Case Company still operated with a more traditional release process. The process was characterized by long testing periods, big and infrequent release phases, and extensive release notes consisting of various reports and approvals from people all over the organization. However, this process did not fit well with the new agile approach to software development, where finished software was produced on a more regular basis and therefore ended up waiting to be released. It is worth mentioning that the organization now operates with a release process that better accommodates this way of developing software. For example, the System Development Department has implemented functionality in their software development system which automatically generates a release note when a new release is done. This enables the department to rapidly release new code into production when necessary, while simultaneously making sure the required level of documentation is in place.

The Application Developer Manager mentioned during the interviews that these changes to the organization's release process also brought along some challenges related to controllability and accountability regarding the release of new software. What types of changes should now be approved by management, and which could be directly released into production? Also, who is now accountable for potential problems caused by the release of new software? On the one hand, management wants to have full insight into all changes made to the production systems, while the development teams do not want to be crippled by unnecessary meetings and formalities that hinder their productivity.

These challenges are often closely related to the organizational culture of the company (C1). The organization has had these processes and routines since they began developing and releasing their software, and might, therefore, be hesitant to change them (C17). It is also possible that they had not considered how this might affect the System Development Department when they started using agile development processes. Organizational culture may also force the System Development Department to interact with business through more traditional practices (C5). The presence of organizational culture as a challenge for agile implementations is also discussed in section 5.2.5.

A consequence of having this type of release process with agile software development teams is that the organization will never take full advantage of the various benefits of agile software development. Agile software development makes it possible to more quickly adapt to changes and develop software that creates value for the customer, but a traditional release process may result in more time before changes are deployed to production and less value for the customer. Another consequence is that some people in the organization may have to spend time on things they probably did not have to, for instance approving releases.

The consequences of traditional release processes may be reduced by transferring some of the responsibility and accountability of releases to the software developers. As it is the System Development Department that has the best knowledge and understanding of which changes that potentially can affect other systems in production, they should be able and allowed to make these decisions on their own. They should not be required to get permission from someone further up in the organization that may have minimal knowledge of what the outcome of the decision will be. Letting the development teams make decisions is also emphasized in the Scrum framework, where they are encouraged to be self-organizing (see section 2.4.1).

5.2.9 Sprint Review

Sprint Review is an important part of Scrum, and even though the System Development Department held them regularly before, they are no longer facilitating these meetings. For certain projects or changes, they have Sprint Reviews, but this is mostly just because some individuals are invested in their systems. It might not always be useful to hold Sprint Reviews, but the consequences of not holding them are present in the Case Company. Some projects have status meetings, but these meetings may not have the same purposes as Sprint Reviews.

One reason for not holding Sprint Reviews might be that there are no common standard or any guidelines for how the System Development Department should operate. As such, there is nothing that says they have to hold Sprint Reviews. Furthermore, there is also currently no dedicated Scrum Master (C13) that can facilitate the Sprint Reviews, which may often lead to no one taking the initiative to hold them. Another reason may be that

the people responsible for projects or changes do not want to, or maybe do not care enough, to be a part of Sprint Reviews. A possible reason is that the department does not see the value in holding Sprint Reviews, and as such think of them as time wasters. This mentality may suggest that the department lack sufficient knowledge of agile values (C15), or that the contents of these types of meetings are covered through other interactions. This can also be characterized as an anti-pattern (C22, see section 2.5), as it is a deviation from Scrum theory.

There might be several potential consequences of not holding Sprint Reviews regularly. As Sprint Reviews are an opportunity for development teams and the stakeholders to meet and collaborate on how to optimize value and plan for future Sprints, choosing not to arrange these types of meetings may result in the Scrum Team not utilizing each others knowledge to the fullest in order to maximize value. Sprint Review meetings are also a place for all the involved parties of the project to jointly evaluate and review how potential use of the product, as well as the marketplace, might have changed during the current Sprint (Schwaber & Sutherland, 2017). Those potential changes could make adjustments to the product or project as a whole necessary. Not organizing Sprint Reviews may, therefore, result in the final product not following the various trends in the market, and in worst case scenario not being competitive compared to other products in the market.

Possible solutions to the challenge may be to teach and show the people responsible for projects or changes, and the development teams, the value of Sprint Reviews. This can help as both parties will be interested in the meetings and their outcome. Hiring a new Scrum Master that is strict when it comes to following Scrum will also make sure the Sprint Reviews are held, but like the first solution, this one may also turn out negative.

5.3 Summary of Findings

This section contains a summary of the challenges discussed in the previous section, as well as their associated causes and consequences. Similar challenges found in the systematic literature review, as presented in table 2.1, are also included. The summary is presented in table 5.2.

Challenge	*Caused by*	*Consequences*	*ID*
Sprint Workload	Imprecise estimates Not breaking down PBIs Unforeseen tasks	Delays in releases Exceeding budgets Testing moved to next sprint	C10
Testing in the Next Sprint	Too much to do in each sprint Code and test in separate PBIs	Development delays and disruptions Rework Unsatisfactory burn down charts	C15, C22
PBI Descriptions	No company-wide standard High level of uncertainty Lack of technical knowledge Too lenient with detailing Suboptimal communication No dedicated Scrum Master	Prolonged development times Ambiguity of what is to be done Unsatisfactory results Budget and time overruns Uncertain what to test	C6, C7, C13, C15, C16,
Business Agility	Organizational culture	Slower time-to-market	C1, C12, C15, C17
Documentation	Misunderstanding of agile values External pressure to deliver functioning software Lack time Organizational culture Limited personal benefit	Additional work Lack of control Waste of time and resources	C1, C15
PBI Grooming	Grooming is deprioritized No dedicated Scrum Master	Specialization Person Dependent	C6, C13, C15
Team Improvement	No sprint retrospective Complacent No time for improvement No dedicated Scrum Master	Specialization No reaching full potential Less motivation for improving	C13, C15, C22
Release Process	Organizational culture	Longer time-to-release Less value customer value Waste of time and resources	C1, C5, C17
Sprint Review	No common standard No dedicated Scrum Master	Lack of collaboration Less customer value Non-competitive products	C13, C15, C22

Table 5.2: Summary of findings from the discussion.

As shown in table 5.2, a total of nine challenges related to agile and its implementation in the System Development Department at the Case Company were uncovered. A wide range of different causes to the challenges was discussed, and some of them are repeating and part of broader challenges. Organizational culture is one of these causes that can be linked to multiple challenges, and it is also one of the related challenges (C1) that has been found in the literature. Several different consequences of these challenges and how they might impact both the System Development Department and

the Case Company were also discussed. Specialization and delays are two important ones.

Even though the challenges uncovered in this thesis have different formulations compared to those found in the strategic literature review by López-Martínez et al. (2016), many of them are, however, closely related. For instance, some of the challenges from the literature review are often related to several of those found in this case study, as they are more overarching and general challenges. These overarching challenges are C13 (availability of trained personnel) and C15 (lack of understanding of agile values). Due to the temporary situation of the Case Company not having a dedicated Scrum Master, one can conclude that C13 is an overarching challenge which can result in poor PBI descriptions, poor PBI grooming, minimal team improvement and lack of Sprint Review. C15 might also be characterized as an overarching challenge since several of the challenges uncovered in the case study are related to deviations from the Scrum framework and agile principles, which hinders the System Development Department from reaching its full potential. Conducting Sprint Reviews and Retrospectives regularly, as well as prioritizing sufficient grooming of PBIs may yield benefits to the System Development Department.

The majority of the challenges and causes uncovered in this thesis are already prevalent in existing theory. However, there was one finding that has not yet been given much attention. This finding was related to how the development teams integrate the responsibility of maintaining their existing system into their Sprints. Due to a high level of uncertainty regarding the amount of maintenance work for a given Sprint, the development teams allocates a set amount of time for this type of work, which sometimes is less than what is needed. Not allocating enough time for maintenance work may lead to several unfortunate consequences, such as lower-prioritized PBIs being pushed to the next Sprint, slower progress on ongoing projects, and unsatisfactory burndown charts. Even though work estimation is a relatively known challenge, it has yet not been given much attention in this situation with varying amount of maintenance work. This type of estimation is also something not accounted for in the Scrum framework, and might, therefore, be a subject for future research.

Chapter 6

Conclusion

As agile approaches to software development have been widely adopted by organizations over the last decades, they have also brought along various challenges. To enrich existing literature on the given topic, this thesis has addressed challenges with agile and its implementation in a System Development Department. This has been done through a qualitative and descriptive case study, were six interviews with employees of different roles within the System Development Department at Case Company were conducted. These interviews were also partly bolstered up by observations from different types of status and planning meetings. During the interviews and observations, nine challenges were uncovered. A discussion and analysis of the potential causes and consequences of these challenges have been provided. The challenges were compared to findings from a strategic literature review, as well as other studies related to challenges with agile implementation.

One of the main findings of the case study is that the System Development Department has not been 100 % true to the Scrum framework, and as such, it is advised that they do a more thorough implementation of the framework. Conducting Sprint Reviews and Retrospectives regularly, as well as prioritizing sufficient grooming of PBIs may yield benefits to the department. Furthermore, the temporary situation of the department not having a dedicated Scrum Master also supports this finding. Another main finding is that organizational culture is one of the causes behind multiple challenges, and these challenges are mostly related to situations where the department and the rest of the Case Company interact.

A contribution of this thesis is that it further supports the findings of other studies related to challenges with agile and its implementation. However, this thesis also discusses a challenge that has not yet been given much attention in existing theory. Given that the development teams are responsible for maintaining their existing system, estimating the amount of maintenance work during the Sprint Planning meeting may be challenging due to inherent uncertainty. An estimation that is too low may lead to several unfortunate consequences, such as lower-prioritized PBIs being pushed to the next Sprint, slower progress on ongoing projects, and unsatisfactory burndown charts. Even though estimation is a relatively known challenge, it has yet not been given much attention in this given situation. It is also something that is not accounted for in the Scrum framework.

6.1 Limitations

A limitation of this thesis is the research method used. It is hard to make generalizations from a single case study, so doing a multi-case study with system development departments from other companies might lead to more generalizable findings. Interviews were the primary method for collecting data, and the usage of more methods might have led to better triangulation and more trustworthy data. Time itself has also been a limitation. More challenges might have been uncovered if there was more time to also interview the business side of the Case Company. Furthermore, conducting a new round of interviews with some of the interviewees might also have revealed additional challenges.

Another limitation is caused by the lack of a dedicated Scrum Master, which might have created challenges that otherwise would not exist. So the results could have been different if the same case study was conducted once a new Scrum Master was in place. The thesis is, therefore, limited by the fact that it also includes challenges that might be temporary.

Since there was no opportunity to interview software developers from the .NET developer team, one can argue that this is a limitation, as additional challenges might have been uncovered.

The last limitation regards the challenges that were discarded due to not being related to agile or its implementation. It is not possible to say for certain if the discarded challenges were related or not, and as such challenges that potentially were related might have been discarded.

6.2 Future Work

A subject that might be applicable for future research is related to the challenge of estimating the amount of maintenance work that may appear during an upcoming Sprint. As this is not accounted for in the Scrum framework, we suggest future research on how this can be managed in a more structured and predictable manner.

References

Agile Alliance. (n.d.-a). *Pair programming.* Retrieved from https://www.agilealliance.org/glossary/pairing/

Agile Alliance. (n.d.-b). *What is Agile Software Development.* Retrieved 3 April 2019, from https://www.agilealliance.org/agile101/

Ali, A. M., & Yusof, H. (2011). Quality in qualitative studies: The case of validity, reliability and generalizability. *Issues in Social and Environmental Accounting*, *5*(1/2), 25–64.

Baxter, P., & Jack, S. (2008). Qualitative case study methodology: Study design and implementation for novice researchers. *The qualitative report*, *13*(4), 544–559.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... Thomas, D. (2001). *Manifesto for Agile Software Development.* Retrieved 3 April 2019, from http://agilemanifesto.org/iso/en/manifesto.html

Bloch, M., Blumberg, S., & Laartz, J. (2012). Delivering large-scale it projects on time, on budget, and on value. *Harvard Business Review*, 2–7.

Cockburn, A. (2002). *Agile software development.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Cohn, M. (n.d.). *User stories.* Retrieved from https://www.mountaingoatsoftware.com/agile/user-stories

Cohn, M. (2015). *Product backlog refinement (grooming).*

Conboy, K., Coyle, S., Wang, X., & Pikkarainen, M. (2011, July). People over process: Key challenges in agile development. *IEEE Software*, *28*(4), 48–57. Retrieved from http://dx.doi.org/10.1109/MS.2010.132 doi: 10.1109/MS.2010.132

Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software*, *119*, 87

- 108. Retrieved from http://www.sciencedirect.com/science/article/pii/S0164121216300826 doi: https://doi.org/10.1016/j.jss.2016.06.013

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). *A decade of agile methodologies: Towards explaining agile software development.* Elsevier.

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50*(9), 833 - 859. Retrieved from http://www.sciencedirect.com/science/article/pii/S0950584908000256 doi: https://doi.org/10.1016/j.infsof.2008.01.006

Easterby-Smith, M., Thorpe, R., & Jackson, P. R. (2015). *Management and business research - 5th edition* (Paperback ed.). Sage Publications Ltd.

Eloranta, V.-P., Koskimies, K., Mikkonen, T., & Vuorinen, J. (2013). Scrum anti-patterns – an empirical study. In *2013 20th asia-pacific software engineering conference (apsec)* (Vol. 1, pp. 503–510).

Flyvbjerg, B., & Budzier, A. (2011). Why your it project may be riskier than you think. *Harvard business review*, *89*(11).

Gregory, P., Barroca, L., Sharp, H., Deshpande, A., & Taylor, K. (2016). The challenges that challenge: Engaging with agile practitioners' concerns. *Information and Software Technology*, *77*, 92–104.

Hsieh, C.-e. (2004). Strengths and weaknesses of qualitative case study research. *University of Leicester Publishing*.

Jacobsen, D. I. (2015). *Hvordan gjennomføre undersøkelser?: innføring i samfunnsvitenskapelig metode.* Oslo: Cappelen Damm Akademisk.

Kniberg, H. (2015). *Scrum and xp from the trenches.* Lulu.com.

Larson, E. W., & Gray, C. F. (2010). *Project Management: The Managerial Process* (5th ed.). McGraw-Hill Irwin.

López-Martínez, J., Juárez-Ramírez, R., Huertas, C., Jiménez, S., & Guerra-García, C. (2016). Problems in the adoption of agile-scrum methodologies: A systematic literature review. In *2016 4th international conference in software engineering research and innovation (conisoft)* (pp. 141–148).

Lorber, A. A., & Mish, K. D. (2013). How we successfully adapted agile for a research-heavy engineering software team. In *2013 agile conference* (pp. 156–163).

Mathiassen, L., & Pries-Heje, J. (2006). Business agility and diffusion of information technology. *European Journal of Information Sys-*

tems, *15*(2), 116-119. Retrieved from https://doi.org/10.1057/palgrave.ejis.3000610 doi: 10.1057/palgrave.ejis.3000610

McCusker, K., & Gunaydin, S. (2015). Research using qualitative, quantitative or mixed methods and choice based on the research. *Perfusion*, *30*(7), 537-542. Retrieved from https://doi.org/10.1177/0267659114559116 (PMID: 25378417) doi: 10.1177/0267659114559116

Moe, N. B., & Dingsøyr, T. (2008). Scrum and team effectiveness: Theory and practice. In *International conference on agile processes and extreme programming in software engineering* (pp. 11–20).

Popli, R., & Chauhan, N. (2013, March). A sprint-point based estimation technique in scrum. In *2013 international conference on information systems and computer networks* (p. 98-103). doi: 10.1109/ICISCON.2013.6524182

Prause, C. R., & Durdik, Z. (2012). Architectural design and documentation: Waste in agile development? In *2012 international conference on software and system process (icssp)* (pp. 130–134).

Rahmanian, M. (2014). A comparative study on hybrid it project management. *International Journal of Computer and Information Technology*, *3*(05), 1096–1099.

Rigby, D. K., Sutherland, J., & Takeuchi, H. (2016). Embracing agile. *Harvard Business Review*, *94*(5), 40–50.

Schwaber, K. (2000). *Agile project management with scrum.* Pearson Education.

Schwaber, K., & Sutherland, J. (2017). The scrum guide. *Scrum Alliance*.

Scrum Institute. (n.d.). *Scrum burndown chart.* Retrieved from https://www.scrum-institute.org/Burndown_Chart.php

Scrum.org. (n.d.). *The scrum framework poster.* Retrieved from https://www.scrum.org/resources/scrum-framework-poster

Serrador, P., & Pinto, J. K. (2015). Does agile work? — a quantitative analysis of agile project success. *International Journal of Project Management*, *33*(5), 1040 - 1051. Retrieved from http://www.sciencedirect.com/science/article/pii/S0263786315000071 doi: https://doi.org/10.1016/j.ijproman.2015.01.006

Stoica, M., Mircea, M., & Ghilic-Micu, B. (2013). Software development: Agile vs. traditional. *Informatica Economica*, *17*(4).

Theocharis, G., Kuhrmann, M., Münch, J., & Diebold, P. (2015). Is water-scrum-fall reality? on the use of agile and traditional development

practices. In *International conference on product-focused software process improvement* (pp. 149–166).

VersionOne. (2012). *7th annual state of agile survey.* Retrieved from `https://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf`

West, D., Gilpin, M., Grant, T., & Anderson, A. (2011). Water-scrum-fall is the reality of Agile for most Organizations today. *Forrester Research*, *26*.

World Economic Forum. (2018). *Annual report 2017-2018.* Retrieved from `https://www.weforum.org/reports/annual-report-2016-2017`

Yazan, B. (2015). Three approaches to case study methods in education: Yin, Merriam, and Stake. *The qualitative report*, *20*(2), 134–152.

Yin, R. (2017). *Case study research and applications: Design and methods.* SAGE Publications. (E-BOOK)

Appendices

A Interview Protocol

Intervjuguide

Oppgave: Se på hvilke problemer som fremdeles eksisterer i en bedrift som har gått fra tradisjonell til smidig utvikling.

- **Hvilken funksjon / rolle / ansvar har du I Santander?**
- **Hvor lenge har du jobbet I Santander?**
- **Hvordan opplever du samspillet med den øvrige delen av bedriften?**

- Hvordan jobber utviklingsavdelingen? Prosessflyt?
  - Hvorfor har dere valgt dette?
  - Har dere alltid jobbet sånn?
  - Hvordan er rolleansvarsfordelingen?
- Hvorfor gikk dere over til å jobbe smidig? (Hvorfor har dere endret måten dere jobber på?)
  - Hvordan opplever du dette?

- Hvordan er prosessen når et nytt "prosjekt" settes I gang?
  - Hvem kommer med requirements?
  - Er de som kommer med requirements med videre I prosessen?
  - Hvem blir Product Owner?
  - Jobber de med scrum-teamet?
- Hvordan er oppfølgingen underveis fra de som kom med "prosjektet"?

- Er testing en integrert del av hver sprint, eller foregår det I etterkant?
  - Hva synes du om denne måten å gjøre det på?
- Hvor fort kommer ferdig utviklet kode ut I produksjon?
- Hvor ofte har dere release av kode?
  - Hva er den største bottleneck / hva bestemmer når dette skjer?

- Dere jobber etter smidig-prinsipper, I hvor stor grad gjør resten av bedriften det?
  - Følger de dere forholder dere til også disse prinsippene?
  - Er det et mål / meningen at de dere jobber med (eller hele bedriften) skal jobbe etter Agile-prinsippene, eller er kun hovedsakelig begrenset til systemutvikling?

- Er det noe du føler som gjenstår for at utviklingsavdelingen skal være fullstendig smidig?

- Hva vil du si er de største utfordringene knyttet til måten dere jobber på?
  - Hvordan jobber dere for å bli bedre på dette?

UiA University of Agder
Master's thesis
Faculty of Engineering and Science
School of Business and Law