NTNU – Trondheim
Norwegian University of
Science and Technology

Coordination in Large-Scale Agile Development

Ragnar Alexander T Morken

Master of Science in Informatics
Submission date: May 2014
Supervisor: Maria Letizia Jaccheri, IDI
Co-supervisor: Torgeir Dingsøyr, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

In the last decade agile software development methods has become one of the most popular topics within software engineering. Agile software development is well accepted in small projects among the practitioner community and in recent years, there has also been several large-scale projects adopting agile methodologies, but there is little understanding of how such projects achieve effective coordination, which is known to be a critical factor in software engineering.

This thesis describe an exploratory case study on how practices in an agile large-scale software development projects impact coordination. The goal is to provide a rich description on how practices are implemented in large-scale projects, and how they contribute to achieving effective coordination.

The main findings are that agile practices are implemented in the same fashion as they are in small projects, and effective coordination is achieved by reducing the needs for cross-team coordination through non-agile practices and handling unforeseen cross-team dependencies with agile practices.

Keywords: Large-scale, Coordination, Agile, Scrum, Test driven development, Critical path method, Software development, Software engineering

Sammendrag

I det siste tiåret, har smidig systemutviklingsmetoder blitt et av de mest populære emnene innen systemutvikling. Smidig systemutvikling er godt akseptert i systemutviklingsmiljøet, og i senere år har også store prosjekt tatt i bruk smidig utvikling, men det er lite forståelse om hvordan slike prosjekt oppnår effektiv koordinering, som er vel annerkjent som en kritisk faktor i systemutvikling.

Denne oppgaven beskriver en utforskende case study om hvordan praksiser i et stor-skala smidig systemutviklingsprosjekt påvirker coordinering. Målet er å gi en innholdsrik beskrivelse om hvordan disse praksisene er implementert i stor-skala prosjekt, og hvordan de bidrar til å oppnå effektiv koordinering.

Hovedfunnene i denne oppgaven er at smidige praksiser er implementert på samme måte som i mindre prosjekt, og at effektiv koordinering oppnås ved å redusere behovet for koordinering på tvers av teamene gjennom ikke-smidige praksiser, og deretter håndtrere de uforutsette avhengighetene på tvers av teamene gjennom smidige praksiser.

Nøkkelord: Stor-skala, Koordinering, Smidig, Scrum, Testdrevet utvikling, Kritisk sti metode, Programvareutvikling, Systemutvikling

Preface

This report was written as the final step in a 2 year Master program in Informatics at the Norwegian University of Science and Technology (NTNU). The author would like to thank all the interviewees for letting him interview them and the project management for letting him conduct the study on the project. Without them, he wouldn't have been able to write this thesis. They took time out of a busy schedule in an important phase of the project to provide valuable data for this study. Thank you!

The author would also like to thank his supervisor for this thesis, Torgeir Dingsøyr. Due to a technical formality at NTNU, he is listed as the co-supervisor in the documents at NTNU, but has, for all intents and purpose, functioned as the primary supervisor. He has spent a lot of time with the author for more than a year an a half, discussing the thesis and research plan. Through a constant positive attitude, he has helped the author through discussions, coaching and continuous feedback, which has been invaluable to the author. This thesis would not become true without his great supervision. Thank you so very much, Torgeir!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem description

Computer systems developed in the 1970s was designed primarily to complete a single task e.g. a payroll system. Todays systems are usually a lot more complex, which leads to a more complicated development process. In addition to the increasing complexity of system development, the cost of system development is fairly high. In addition to the complexity of the projects, the technology is changing rapidly.

Agile software development methods are particularly designed to deal with change and uncertainty. They favor intensive face-to-face communication and other apparently simple practices. In 2013, the Norwegian Labour and Welfare Administration (NAV) stopped a project after spending 700 million Norwegian kroner on the development project. It is unknown how much resources spent on development is currently usable. The NAV project is one out of several projects that have been cancelled. Rubinstein (2007) estimated that 19% of all software projects were cancelled, and 46% were over time or over budget in 2006. Research done by Flyvbjerg and Budzier (2011) shows one out of six projects studied had a cost overrun of 200%. Not only is the costs related to development high, but modern information systems are now integrated in so many aspects of the organisation that the development of information systems pose a singular new risk for the organisation as a whole. The German company Toll Collect (a consortium of Daimler-Chrysler, Deutsche Telekom, and Confiroute of France) had an estimated loss of $10 billion in lost revenue due to the developers struggled to combine the different software systems.

There are several factors leading to the success or failure of a project, but effective coordination has been acknowledged, long before the arrival of agile software development, as a critical component in organisations generally (Curtis et al. (1988) and Van de Ven et al. (1976)). Coordination has also been particularly emphasised as a critical factor in software development (Kraut and Streeter (1995) and Strode et al. (2012)).

The enormous costs related to failed system development projects and the critical impact effective coordination has on the failure or success of a project, substantiates the importance of research in the field of coordination in large-scale agile software development.

1.2 Initial research question and significance

When constructing a research question, there are several things to keep in mind. It should be interesting for both the reader and the researcher and there must be a balance so that it is not too open, nor too specific. It should be a topic where the researcher is at least somewhat invested, and maybe most important, it should be a problem that is actually worth studying. The first step towards defining a research question, was through conversations with the supervisor. These conversations made it easier to see both the possibilities and limitations of the thesis.

The authors interest for project management as well as the ability to draw from the supervisors research and knowledge within the field of agile development was a big factor into deciding what would be studied. Finally, through these conversations, the topic of coordination in large-scale projects using agile development emerged. The research question used in this thesis is:

How does practices used in large-scale agile development affect coordination?

Further reasoning behind the research question can be found in chapter 4.1.

To conduct this study, the author was given the opportunity to do a case study on an agile large-scale project, developing a community critical system. An initial literature review was conducted, which proved valuable for the author in order to better understand the challenges of scaling up the agile development method, and to place the thesis in context of what has already been published.

1.3 Scope of the report

Several important issues and potential research areas arise when studying large-scale agile development, such as how does management operate compared to traditional large-scale project management, how does large-scale agile development compare to small-scale agile projects, how does agile development compare to traditional development in large-scale projects and how can/does agile development take place large-scale agile development projects. These are all relevant research areas, so in order to limit the scope of this study, this thesis will focus on coordination in large-scale agile development projects. Through the initial literature review it was clear that there is little research done on agile development in large-scale projects, and the research done on coordination in agile development was rather scarce.

The outcome of this thesis should not be a new theory, but rather insight and a thorough description of how coordination is done in large-scale agile projects. As both coordination agile development in themselves are large research areas, this thesis will limit to looking at how the practices used in large-scale agile development project affects the coordination. As each practice in it's own could be a research study in themselves, the scope of this project is limited to looking at how they impact coordination.

In terms of coordination, this thesis focus on the coordination across teams, rather than within the different teams. The research question in this theses will be covered later (see section 4.1). The thesis will also give a brief introduction to software development methodologies. Since development methodology in practice often differs from the pure theory based textbook examples, traditional, agile and lean methodologies will be presented. In addition, there will be a short introduction into the field of coordination in software development. This is to give the reader of the report a better understanding of the theory behind the research, and to put the discussion into context for the reader. The chosen content of the theory chapters, is a result of what is being used in the case studied.

1.4 The target audience of the report

Agile development is still rather young, but is rapidly emerging. Therefore it will hopefully have some value for the following audiences:

- **Computer science students** might find it interesting to see how agile methodologies are done in large-scale projects, since most of the textbooks focus on smaller projects. It can also be interesting to see how a theoretical framework is in a real life setting, in contrast to an educational setting.

- **Researchers** in the field of computer science or coordination may find it interesting to read a case study on a field with rather scarce information.

- **Practitioners** who are interested in improving or adopting agile methods to a large-scale projects, can possibly gain valuable insight in how it is done in a large-scale project, as well as what adaptations have been made to make an agile methodology work in a large-scale project.

- **Customers of software development** may value a neutral third party's view upon how agile development works in practice in a large-scale project.

1.5 Report structure

The thesis is written with the following report structure:

Chapter 1 - Introduction: An introductory chapter that gives a brief overview of the report and its structure, the scope of the report and describes the target audience of the report.

Chapter 2 - Software development and coordination: This chapter is where all the theory related to software development methodologies are presented. It contains a description of both the concept of agile development, as well as some information about specific relevant methodologies and practices.

Chapter 3- Coordination: A theory based chapter that gives the reader relevant introduction to coordination and coordination within the field of software development.

Chapter 4 - Research design This chapter describes the theory and choices behind the research method and describes how it is designed and conducted.

Chapter 5 - Results This chapter presents the results that were collected during the study. The results are grouped according to the grouping presented in section 4.6

Chapter 6 - Discussion This chapter discusses the results found, other issues that are relevant for the discussion, and the validity of this thesis.

Chapter 7 - Conclusion This chapter presents the conclusion, further work and possible improvements.

Chapter 2

Software development

When a software system is being developed, the methodology refers to the chosen framework that is used to plan, structure and control the development process. The goal of this chapter is to present relevant research and theory to software development methodologies and practices, in order to establish a foundation for the research design and the following discussion. The first section will describe the historical evolution of software methodologies in order to better understand the foundation of the modern software development methodologies. After that, different methodologies and concepts that are relevant to the research will be presented, in order to provide context for the reader. Finally, existing guidelines, research and some challenges related to scaling up the agile methodologies to a large-scale project will be presented.

2.1 Historical evolution of software development methodologies

According to Avison and Fitzgerald (2003), software was created up to the 1960s without any kind of defined software development process. In the 1960s, the software engineering as a discipline emerged, and has in the following years grown in both scale and importance. Avison and Fitzgerald (2003) has divided and defined 4 eras of software development methodology: *pre-methodology, early methodology, methodology* and *post-methodology era.*

Pre-methodology era

In the pre-methodology era, around 1950s and 1960s, the emphasis was on the programming and problem solving, often coupled tightly with hardware constraints. The development was done without any formalised processes, and too much responsibility without any control was put in the hands of individuals. The developers were highly technical skilled, but had a negligible communication with the stakeholders and a low understanding of the business aspect of the company. This led to several problems such as difficulty meeting business needs, inflexibility and dissatisfied users.

Early methodology era

In the 1970s and 1980s, there was a high focus on identifying the phases and stages of the development process, in order to increase management of software development. The system development life cycle, better known as the waterfall model, which is an extension of traditional project development life cycle, was introduced in this era.

Methodology era

The 1980s and 1990s is considered the methodology era. This era is in many ways a response to the issues and limitations that the waterfall model had. Several approaches emerged with a collection of recommended rules, techniques, tools, documentation and management principles. Although these approaches were rather different, their motivation were the same. Achieving better products, improve the development process, and standardize the methodology.

Post-methodology era

The post methodology era lasted from the 1990s to present day, and can be considered the reappraisal of some of the values from the pre-methodology era. Previous eras had brought an overly complex and unrealistic sets of rules and tools, with disappointing productivity. Among several other approaches, this reappraisal brought a focus on incremental methods, which is the foundation of agile software development.

2.2 Traditional software development

The methods from early methodology era, and the methodology era are in modern day considered traditional software development methods. According to Wysocki and McGary (2003) traditional project management divides a project development into 5 steps (See figure 2.1). This model is also considered to consist of four steps plus control, because the third and fourth step are so intertwined.

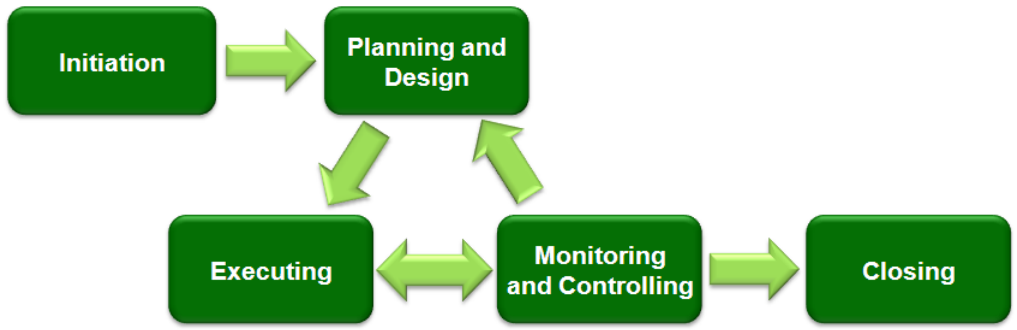


Figure 2.1: Traditional development (Wysocki and McGary (2003))

Initiation: In the first step, the project is defined, a scope for the project is set, milestones are created and the project manager is appointed.

Planning and design: In the second step, the project is broken down into smaller components that are required for the overall success of the project. This process is called a Work Breakdown Structure (WBS). The phase also includes risk analysis and an estimation of resources and time.

Executing: Project execution simply means that the tasks listed in the plan are accomplished in the way they were conceived.

Monitoring and controlling: In the execution phase challenges are bound to come up that the project manager must handle accordingly.

Closing: The project is formally closed by writing a summary of the project.

2.3 Waterfall model

In 1970, dr. Winston Royce adapted the traditional project management methodology to suit software development into a method known as the waterfall method. The waterfall method is typically used in an ordinary large-scale software development project.

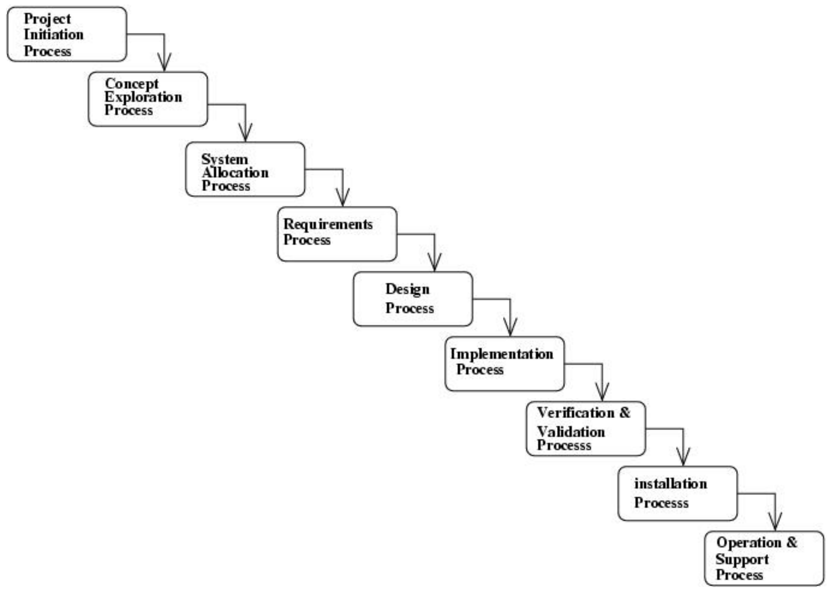

Project Initiation Process
Concept Exploration Process
System Allocation Process
Requirements Process
Design Process
Implementation Process
Verification & Validation Processs
installation Processs
Operation & Support Process

Figure 2.2: The waterfall model (Royce (1987))

A major drawback to this model, which Royce (1987) presents in his paper, is that the steps cannot be separated, and that they had to run in a sequential order. Each phase produces a set of documents and reports, which creates the basis for the next phase, e.g. the system analysis produces an analysis document, which will be the foundation for the program design phase. This means that the entire development process is very little flexible when it comes to requirements and specifications throughout the project lifecycle.

Royce (1987) points out that there is room for feedback, and thereby making the model more flexible, however Hawryszkiewycz (1994) argues the case that this is seldom done, since every step has a single sign-off point where one activity is terminated, thereby shifting the focus over to the next phase.

2.4 Agile software development

There is no widespread agreement on what the concept of agile actually refers to in software development, however it has generated a lot of interest among practitioners and in the academia. The introduction of extreme programming method (better known as XP), has been widely acknowledged as the starting point for the various agile software development approaches. Since then, there has been several methods which have been either invented or rediscovered, that appears to be in the same family of methodologies (Abrahamsson et al. (2002)).

Despite there not being a widespread agreement on what the concept "agile" actually refers to in software development, there have been several authors who have tried to define agile development. In this master thesis, the author has chosen to use the definition from Nerur and Balijepally (2007) in this thesis:

"The traditional mechanistic worldview is today being challenged by a newer agile perspective that accords primacy to uniqueness, ambiguity, complexity, and change, as opposed to prediction, verifiability and control. The goal for optimization is being replaced by flexibility and responsiveness."

Agile software development is in many ways a reaction to the more traditional development. A core differences is how a problem is approached. In traditional development, a problem is approached as fully specifiable, and that optimal and predictable solutions exist for every problem. Problems are therefore solved through extensive planning and a high degree of reuse to make development as efficient and predictable as possible.

Agile development on the other hand sees problems in a constantly changing and unpredictable environment, where one relies a high degree on people and their creativity, rather than on process.

Agile development is also a subset of iterative and evolutionary software development methods. Larman (2004) writes that iterative development is a method, where the project life cycle consists of several iterations, where every iteration ends up with a deliverable piece of software. Evolutionary software development means building a software that changes over time. A typical change is new requirements arriving as the project progresses.

2.5 Scrum

Scrum is a software development methodology building on the ideas of agile development. It is an empirical approach applying the ideas of industrial process control theory to systems development resulting in an approach that reintroduces the idea of flexibility, adaptability and productivity (Schwaber and Beedle (2002)). It does not focus on specific techniques for the system development phase, but rather focus on how team members should function in order to ensure production of a flexible system in an environment with constantly changing factors.

The basic concept of scrum is that there are several environmental and technical variables involved in system development, which are likely to change during the development process. This makes the development process unpredictable and complex, which requires the development method to be flexible and adaptive. Therefore, a traditional linear sequential development strategy is ill suited, since it requires a much more stable and predictable environment. Scrum helps to improve the existing engineering practices (e.g. testing practices) in an organization, since it involves frequent management activities aiming at constantly identifying any deficiencies or impediments in the development process as well as the practices that are used (Abrahamsson et al. (2002)). This section will present the scrum guidelines as described by Abrahamsson et al. (2002), starting with the different phases of scrum, before describing the different elements and practices used in scrum.

The different phases

The scrum process includes three phases. The pre-game phase, the development phase and the post-game phase (see figure 2.3).

The pre-game phase: This first phase can be divided into two sub-phases: Planning and Architecture.

- *Planning* includes defining the system being developed. This is done through creating a product backlog list, which contains all the requirements that are currently known. The requirements are here gathered from various origins e.g. sales, marketing, customer support, and are prioritized in order of importance, and the effort needed to develop features that meets these requirements are estimated. Planning also

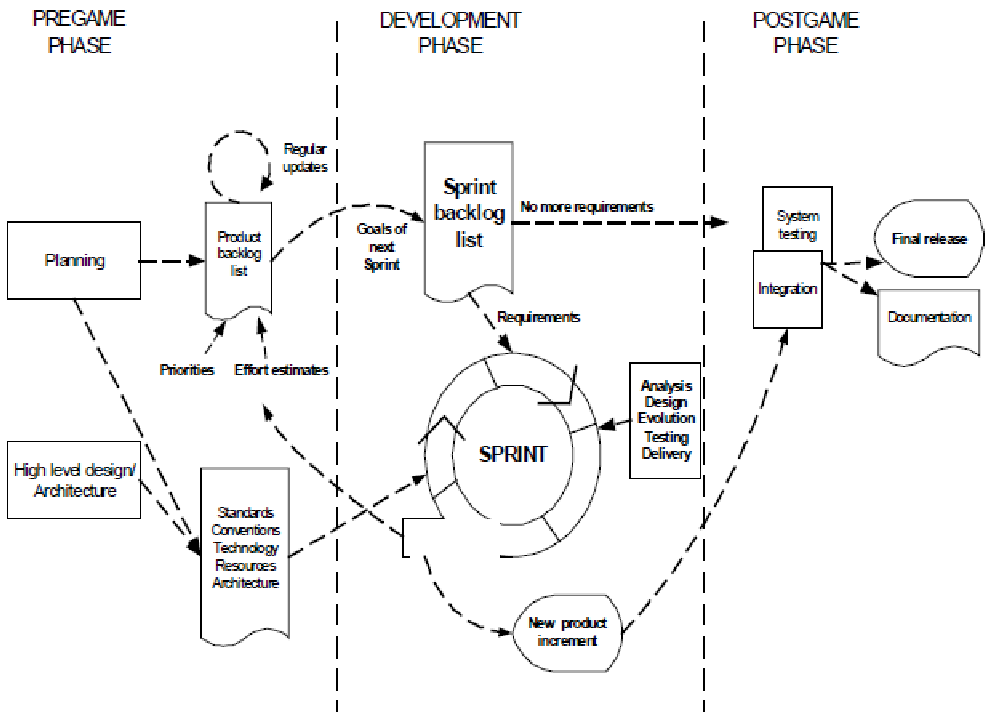


Figure 2.3: The scrum process (Abrahamsson et al. (2002))

includes defining the project team, tools and other resources, risk assessment, controlling, training needs and verification management approval (Abrahamsson et al. (2002));

- The *architecture* part of the pre-game phase, is the high level design of the system, including the architecture which is planned for the system, based on the current items in the product backlog.

The development phase: The development phase is the agile part of Scrum. The development phase is where the product is developed through iterations of sprints (see section 2.5). This is to handle all the unpredictable variables such as time frame, quality, requirement, resources implementation technologies and tools. Rather than taking these matters into consideration only at the beginning of the project, they are continually reevaluated and controlled in order to flexibly adapt to the changes.

The post-game phase: The post-game phase is the final phase of the process. This phase is entered when an agreement has been made that the requirements are completed. In this phase, no more items or issues can be added to the backlog, and no new ones can be invented. The system is now ready for release, and preparation for this should be done during this phase.

Scrum does not require or provide any specific software development methods/practices to be used. Instead, it requires certain management practices and tools in the various phases of Scrum to avoid the chaos caused by unpredictability and complexity (Schwaber (1997)).

Product backlog

The product backlog is an ordered list of everything that is needed in the final product, based on current knowledge, such as features, bug fixes, requirements, functions, defects. It comprises a prioritized and constantly updated list of technical and business requirements for the system being developed. In figure **??** an example of a product backlog is displayed with a description of the task that needs to be done, an estimation of how many hours it takes to complete the task, and who is responsible for doing the task.

Effort estimation

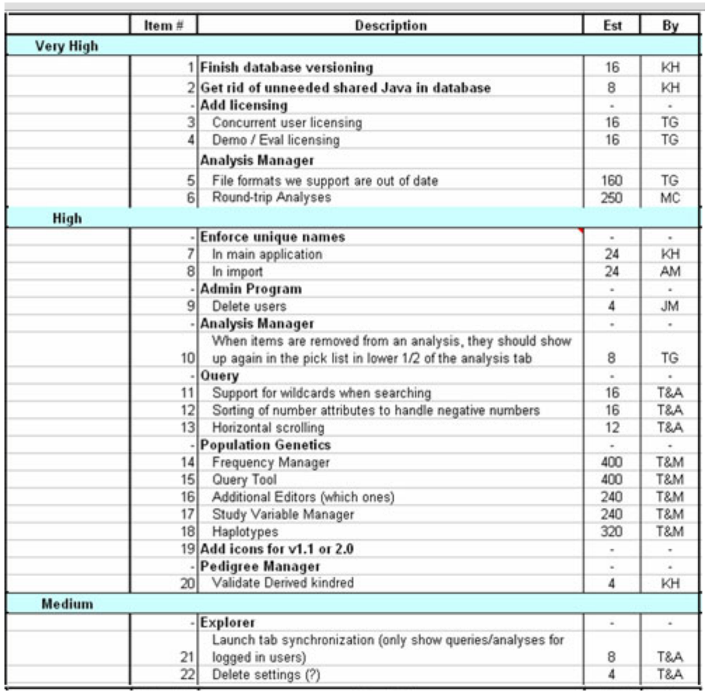
Effort estimation is an iterative process, where the items in the backlog is evaluated in order to give a best possible estimation of how much effort it takes to solve an item, given the information available at the current time.

Sprint

A Sprint is a time period (typically 1 - 4 weeks) where development occurs on a set of backlog items. The Scrum Team organizes itself to produce a new executable product increment in a sprint. The working tools available for the team are sprint planning meetings, sprint backlog and daily Scrum meetings.

Sprint backlog

The sprint backlog is a list of items selected from the product backlog which are to be implemented in the next sprint. Which items are to be included in the sprint backlog is selected by the Scrum Team, Scrum Master and the Product Owner in the sprint planning meeting based on their priority in the product backlog, and the goals set for the sprint. The key difference between the sprint backlog and product backlog is that the sprint backlog is stable until the sprint is completed. When all the items in the sprint backlog are completed, a new iteration of the system is delivered. An example of a sprint backlog can be found in figure 2.4.



	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	**Population Genetics**	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	**Add icons for v1.1 or 2.0**	-	-
	-	**Pedigree Manager**	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	**Explorer**	-	-
	21	Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

Figure 2.4: Example of a sprint backlog[a]

[a]http://epf.eclipse.org/wikis/scrum/Scrum/workproducts/resources/productbacklog.jpg

Sprint planning meeting

The Sprint planning meeting is a meeting organized by the Scrum Master, which is organised into two separate phases. In the first phase, the customers, users, management, product owner and scrum team participate to decide the goals and functionality of the next sprint. The second phase of the meeting is held by the scrum master, and the focus is here on how the product increments is to be implemented during the sprint.

Daily scrum meeting

The daily scrum meetings are held in order to keep track of progress of the scrum team continuously. They are serve as a planning meeting to see what has been done since the last meeting, and what should be done before the next one. If anyone are experiencing problems, or other variable matters, these should be discussed in this meeting. Any deficiencies in the development process are looked for, identified and removed in order to improve the process. The scrum master is responsible for the daily scrum meetings which should be held every day, and last no more than 15 minutes, and everyone participating should be standing through the entire meeting. For this reason the daily scrum meeting is often referred to as a "standup meeting." The scrum master and scrum team attend all the daily scrum meetings, but it is also possible for e.g. management to participate in the meeting as observers.

Review meeting

Sprint review meetings happens on the last day of the sprint. Here the results of the sprint is presented to the management, customers, users and the product owner in an informal meeting. Typically this takes the form of a demo of the new features. The goal is to assess the product increment and make decisions about future activities. The review meeting may bring out new problems, opportunities or other things that are added to the backlog and even change the direction of the system being built.

Retrospective meeting

The retrospective meeting is facilitated by the scrum Master and takes place after the sprint is completed. Here, the team discusses the previous sprint and determines what could be changed in order to improve the next sprint. While the review meeting focuses on what has been done, and what should be done, the retrospective focuses on the process of how it's done.

Figure 2.5: Example of daily scrum meeting / Standup meeting[a]

[a]http://www.xqa.com.ar/visualmanagement/wp-content/uploads/standup2.jpg

Roles and responsibilities

In Scrum, there are six different roles where each role has different tasks and purposes. These are: Scrum Master, Product Owner, Scrum team, Customer and Management. In the subchapters below, the different roles are presented as they are described by Schwaber and Beedle (2002)

- **Scrum Master.** Scrum is facilitated by a Scrum Master, and works as a manager role to ensure that the project is carried out according to the practices, values and rules of *Scrum*, and that it progresses as planned. The Scrum Master works as a link between the other roles by interacting with the project team, the customer(s) and the management during the project. He is also responsible for ensuring that any impediments are removed and changed in the process to keep the team working as productively as possible.

- **Product Owner.** The Product Owner is officially responsible for the project, managing, controlling and making the product backlog list visible. He is selected by the Scrum Master, the customer and the management, and functions as the voice of the customer. The product owner makes the final decision of the tasks related to the product backlog and turns issues in the backlog into features to be developed.

- **Scrum team.** The Scrum team is responsible for developing the product increments at the end of each sprint, and are involved in effort estimation, creating the spring backlog, reviewing the product backlog list and suggesting impediments that needs to be removed from the project.

- **Customer.** The customer participates in the tasks related to product backlog.

- **Management.** Management is in charge of final decision making, as well as standards and conventions to be followed in the project. The management also participates in setting goals and requirements for the project.

2.6 Extreme programming

Extreme programming (XP) is explained by Beck (1999) as an evolution of the traditional development models (see figure 2.6). Due to the long development cycle, the cost of changing a piece of software increased drastic over time. Because of this, it was important to make the biggest, most far reaching decisions as early in the life cycle as possible. This section present the guidelines and theory related to extreme programming. Practices that are particularly relevant for this study are presented in further detail in subsections bellow.

Beck and Andres (2004) writes that XP was "theorized" on the key principles and practices used in a number of successful trials in practice. Although the individual practices of XP were not new, they were collected and lined up in order to function in symbiosis with each other, and thus forming a new methodology within the field of software development. The name Extreme programming comes from taking different principles viewed as common sense, and taking them to extreme levels (Beck and Andres (2004)).

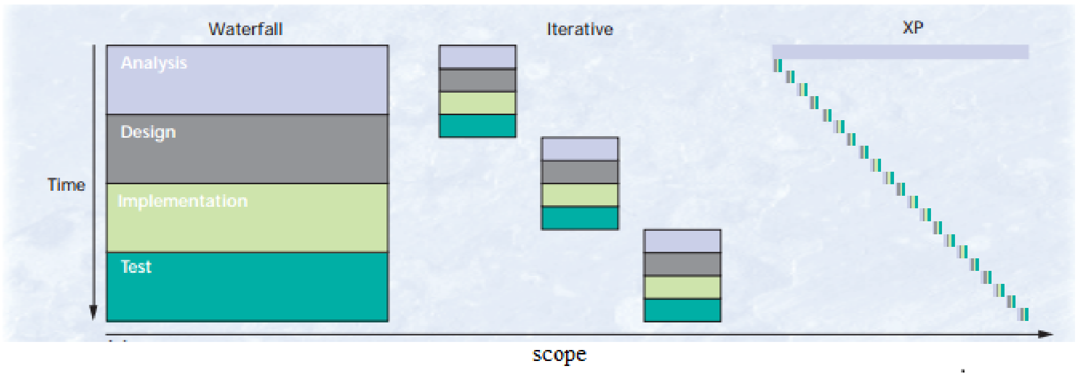


Figure 2.6: The evolution from waterfall to XP as explained by Beck (1999)

Here is a quick summary of each of the major practices in XP as explained by Beck (1999). Those practices that are particularly relevant to this thesis are explained further in a subsection below.

- **Planning game.** The customer decides the scope and timing of releases based on estimates provided by programmers.
- **Small releases.** The system is put into production before solving the whole problem. New releases are made often.
- **Metaphor.** The shape of the system is defined by a metaphor or set of metaphors between the customer and programmers.
- **Simple design.** At every moment, the design runs all the tests, communicates everything the programmers want to communicate, contains no duplicate code, and has the fewest possible classes and methods.
- **Tests.** Programmers write unit tests which all are collected and must run correctly. The customer writes functional tests for the stories in an iteration. See section 2.6.1 for more information
- **Refactoring.** The design of the system is evolved through transformations of the existing design that keeps all the tests running.
- **Pair programming.** All production code is written by two people at one screen/keyboard/mouse. See section 2.6.2 for more information.

- **Continuous integration.** New code is integrated with the current system after no more than a few hours. When integrating, the system is built from scratch and all tests must pass or the changes are discarded.
- **Collective ownership.** Every programmer improves code anywhere in the system at any time if they see the opportunity.
- **On-site customer.** A customer sits with the team full-time.
- **40-hour weeks.** No one can work a second consecutive week of over-time. Even isolated overtime used too frequently is a sign of deeper problems that must be addressed.
- **Open workspace.** The team works in a large room with small cubicles around the periphery.
- **Just rules.** By being part an extreme team, you must sign up to follow the rules. But they're just the rules. The team can change the rules at any time as long as they agree on how they will assess the effects of the change

2.6.1 Test driven development

Test driven programming (TDD), also referred to as test-first(TF) programming, is described by Beck (2003) as a programming technique used in extreme programming where the programmer writes unit tests before he writes any new code. TDD is not a testing technique, but rather a programming and design technique, which by its very nature is iterative. When writing code, the programmer specifies how the program needs to work before he writes the code. After that, the programmer writes the code needed to pass the test. After the test has passed, the programmer looks at the code and if needed, improves the design.

The positive properties of TDD is described by Erdogmus et al. (2005) as follows. Through testing, the programmer gets constant feedback. Writing tests encourages the programmer to decompose problems into smaller manageable tasks, which also has a positive affect on low level design. Frequent tests also ensure a certain degree of quality. Pančur and Ciglarič (2011) writes that there are only a handful of controlled experiments on TDD, and the results have often been inconclusive or conflicting, and is a field of study which needs more research in order to verify or disprove the believed benefits of TDD.

2.6.2 Pair programming

Pair programming is a simple concept where all tasks must be performed by pairs of programmers using only one display, keyboard and mouse. Padberg and Muller (2003) points to two primary benefits with using pair programming.

- A pair of programmer develops software faster. This is called the *Pair speed advantage.*
- The code produced by a pair of programmers has less defect density compared to code written by a single programmer. This is called the *pair defect advantage*

According to Padberg and Muller (2003), the usual reasoning behind the potential advantages are as follows. Pair programming allows the developers to share their ideas and thought immediately. This allows them to come to solutions faster, potentially on a better foundation and it also helps to eliminate defects early. In addition to this, there is an ongoing review of the code by the other programmer continuously, which reduces the defect density.

The primary challenge with pair programming is whether the extra cost in manpower of pair programming is balanced by the potential benefits.
There has been research on the advantages and disadvantages vs cost, but according to Lui and Chan (2006), the results have seemed to lead to contradictory conclusions by different researchers, and Padberg and Muller (2003) concludes that the amount of reliable empirical data is very limited.

2.7 Lean

Poppendieck (2007) describes Lean software development as an application of the Toyota Product Development System to software development. It is a product development paradigm with an end-to-end focus on creating values, optimizing value steams, empowering people and continuously improving.

Lean thinking revolves around waste and value. The goal is to add value in every chain as rapidly as possible, and eliminate waste, where everything that does not directly add value is considered waste.

The method was first described by Tom and Mary Poppendieck when they published a book (Poppendieck and Poppendieck (2003)), where they took the methods described by the Toyota Product Development System, and adapted it to fit software development. In their book, the seven principles used in lean software development are described as follows

- **Eliminate waste**, is in many ways the foundation of lean thinking, and is often referred to as the ”origins of lean thinking”. Nothing should be done that doesn’t add value. Everything that doesn’t add value to the customer should be considered waste.
- **Amplify learning**, also referred to as ”The nature of software development”, means that you should strive for a better understanding and learning of what’s going on in the project. This can be achieved by letting developers work in different phases of the project (design, coding, testing), and keeping the feedback loops with the customer short in order to generate knowledge better and faster.
- **Decide as late as possible**. By making decisions later, more knowledge about the product is obtain, and it is also better suited for changing requirements and environments.
- **Deliver as fast as possible**. Delivering early, gives early feedback, which is a critical factor in software development, and it also adds value to the customer at an early stage.
- **Empower the team**. Bring decisions down to the team level. The people who add value should be allowed to use their potential, and it is important that company management listens attentively to them, and not dismissing them when they are different from their own.
- **Build integrity in**. The systems separate components needs to work well together in a balance between maintainability, efficiency, responsiveness and flexibility. Refactoring and testing are key techniques to ensure integrity, but neither one should be a goal in themselves, but rather a means to an end.

- **See the whole**. Software systems are not simply a sum of their parts, but rather a product of their interactions. Don't optimize parts of the product at the expense of the whole.

Taiichi Ohno of the Toyota production system, identified seven types of manufacturing waste. The seven wastes are presented in 2.1, along with a description of the equivalent waste in software development as described by Poppendieck (2011)

Original description	**Waste in software development**
Overproduction	Extra features
Inventory	Requirements
Extra processing steps	Extra steps
Motion	Finding information
Defects	Defects not caught by tests
Waiting	Waiting, including customers
Transportation	Handoffs

Table 2.1: Seven wastes of software development (Poppendieck and Poppendieck (2003)

2.7.1 Kanban

Kanban originates fro the Japanese words kan(visual) and ban(card/board). It is a method related to lean that also came from the automotive industry at Toyota as a production method (Anderson (2010)). Cocco et al. (2011) describes Kanban as a method where you write work tasks on a card and place them on the Kanban board. As each team member starts on a task, he moves the card over to "in progress". If the task has dependencies, or for some other reason can't be completed, it is moved to the waiting section. When the task is finished, the card is moved over to the completed section. There are also several variations with colour schemes that identifies priorities of tasks, and other section to describe where in the development cycle a task is, such as e.g development, test, deployment. Cocco et al. (2011) writes that this is to make the flow of work visible to all team members, and the *work in process* limits are made explicit on the board. This provides a high visibility to the software process, since it shows the assignment of work to the developers, communicates priorities and makes bottlenecks visible.

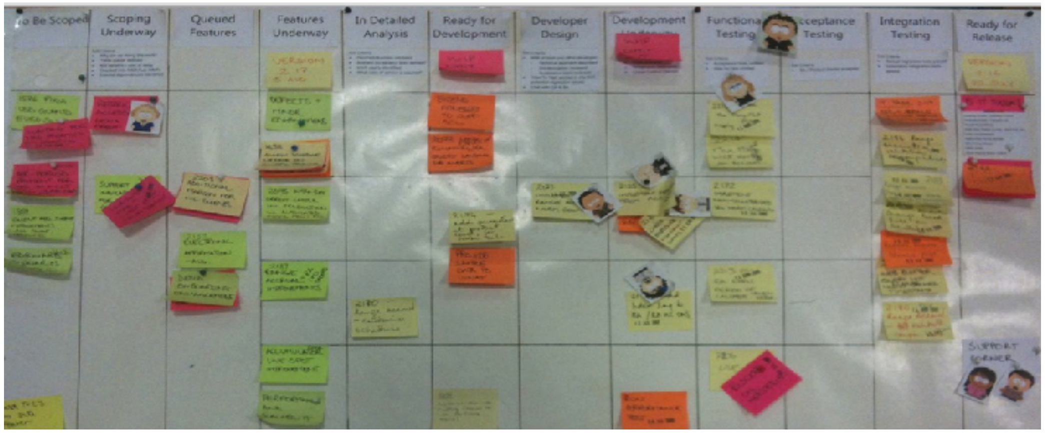


Figure 2.7: Example of a kanban board[a]

[a]http://benjaminmitchell.files.wordpress.com/2011/07/img_0064.jpg

2.8 Large-scale agile software development

This section will first start with describing what is meant by large-scale in software development context, before describing guidelines written in books about how it is possible to scale up agile development to work properly for large projects, before presenting some of the research done on the field. In the initial literature review, the books written about large-scale agile software development(Larman and Vodde (2008) and Eckstein (2004)), were extremely positive to the possibility of scaling up the agile methodologies and practices to work in large projects. Therefore this chapter will also include articles at the end, which are published in magazines, (in contrast to published in scientific journals) in order to show the negative viewpoints of large-scale agile development.

Since both Eckstein (2004) and Larman and Vodde (2008) are very positive to large-scale agile development, software magazine articles have also been included in this chapter in order to represent viewpoints of those who are critical to large-scale agile development.These have been included in this chapter in order describe some of the criticism of large-scale agile development.

The term large-scale development project in itself is rather ambiguous. Large-scale can be defined by the amount of time devoted to the project, by the amount of people working on the project, by how many lines of code the software has. This thesis follows the definition proposed by Dingsøyr et al. (2013), where projects with two or more teams are considered large-scale.

A workshop in 2013 at the International Conference on Agile Software Development (XP2013), discussed the research challenges related to large-scale development. In an article by Dingsøyr and Moe (2013), the discussion is summarized and a suggested research agenda is presented in table 2.2.

Rank	**Topic**	**Description**
1	Inter-team coordination	Coordination of work between teams in large-scale agile development.
2	Large project organisation / Portfolio management	What are effective organisational structures and collaboration models in large projects? How to handle a distributed organisation?
3	Release planning and architecture	How are large projects planned? How can the scope be reduced? What is the role of architecture in large-scale agile?
4	Scaling agile practices	Which agile practices scale and which do not? Why and when do agile practices scale?
5	Customer collaboration	How do product owners and customers collaborate with developers in large-scale projects?
6	Large-scale agile transformation	How can agile practices be adopted efficiently in large projects?
7	Knowledge sharing and improvement	When is the whiteboard not enough? How can communities of practice be established? What measurements are relevant to foster improvement?
8	Agile Contracts	How can contracts change the mindset of the customers from upfront planning to agile principles? What legal limitations exist in contracts that reduce the agility in large projects?

Table 2.2: Suggested research agenda on large-scale agile software development (Dingsøyr and Moe (2013))

Books on large scale agile software development

Although agile development is increasing rapidly in popularity, it is only newly that it has been used in large-scale projects. Both Eckstein (2004) and Larman and Vodde (2008) have written books on large-scale development, but there is a limited amount of peer reviewed research on the topic.

Eckstein (2004) has a large focus on what she refers to as *feature teams*. Each team should have the sufficient competence to deliver whole business functionality. Through the use of feature teams, it is implied that business functionality should never be split across several teams. Taking the whole responsibility for a feature, allows teams to organize themselves and the work.

Larman and Vodde (2008) has a great focus on the core values of agile and scrum when they described how it can be scaled up. The practices used in small projects should still be practised in the separate teams in parallel (see figure 2.8). The primary changes are the sprint planning meetings, the addition of ”scrum of scrums meetings” and a joint retrospective meeting, and they are explained by Larman and Vodde (2008) as follows. The sprint planning meeting is changed by separating the sprint planning meetings in two parts. Part one consists of a meeting with maximum two members per team plus the product owner, where the team representatives self-manage to decide their division of product backlog items. Part two of the sprint is planning is done by each team independently, in the same manner a team would perform a sprint planning meeting in a small project. A representative from team B can however attend team A's planning meeting if there are coordination issues. Team retrospective meetings should be held individually per team, but there should also be a joint retrospective meeting where the scrum masters and one representative from each team meet to identify and plan improvements for the overall product or organisation. Scrum of scrums is an open meeting where all the scrum masters meet with the aim to increase information sharing and coordination.

Larman and Vodde (2008) differs from Eckstein (2004) in the use of feature teams. Larman and Vodde (2008) emphasises that in order to maintain agility across teams, there can not be speciality teams. Most teams should be able to do any product backlog items, which is critical for large-scale agility.

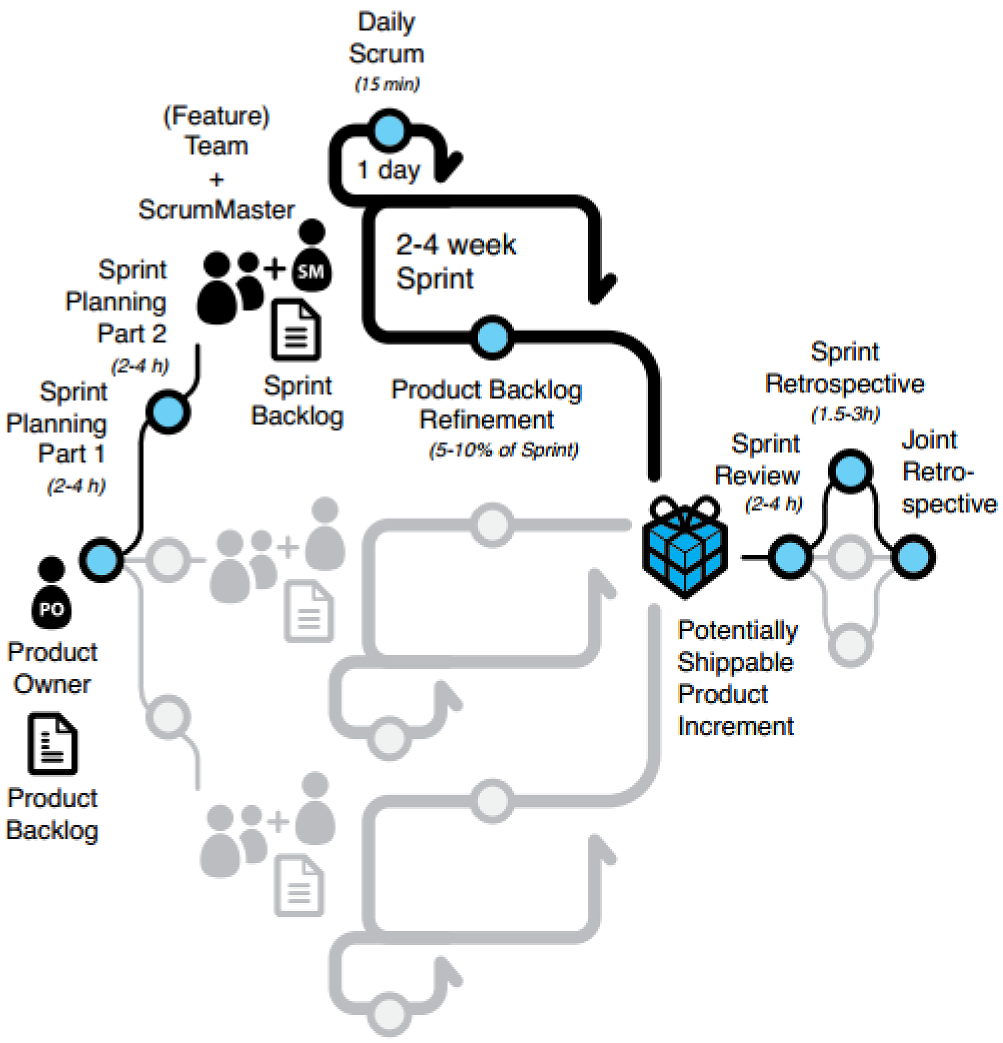

Daily Scrum (15 min)
1 day
(Feature) Team + ScrumMaster
SM
2-4 week Sprint
Sprint Planning Part 2 (2-4 h)
Sprint Backlog
Product Backlog Refinement (5-10% of Sprint)
Sprint Planning Part 1 (2-4 h)
Sprint Retrospective (1.5-3h)
Sprint Review (2-4 h)
Joint Retro-spective
PO
Product Owner
Product Backlog
Potentially Shippable Product Increment

Figure 2.8: Large-scale framework by Larman and Vodde (2008)

Previous research

In a systematic review of empirical studies on agile software development, Dybå and Dingsøyr (2008) writes that one of the most common criticisms of agile development method is that agile development works well for small team, but other processes are more appropriate for larger projects.

In a case study on the impact of continuous release planning in large-scale scrum development organisations by Heikkilä et al. (2013), they found that the benefits were increased flexibility and decreased development lead time, waste was eliminated in the planning process, and increased developer motivation. The challenges were overcommitment caused by external pressure (Mainly because the product management still worked in the old way), managing non-feature specific work and balancing between development efficiency and building generalist teams.

Van Waardenburg and Van Vliet (2013) presents a grounded theory of the challenges of using agile methods in traditional enterprise environments. Projects with enterprise environments are typically confronted with a large and complex IT landscape. The research results are in line with challenges facing smaller companies and projects when going from traditional to agile development.

In a case study conducted by Paasivaara et al. (2008), they look at how agile practices work in a global software development project consisting of 40 people distributed between Norway and Malaysia. In order to communicate, the project used several tools such as web camera, email, chat functions and telephone conferences. The project transitioned from a combination of waterfall and iterative development, to scrum 1.5 years before the study was conducted. Paasivaara et al. (2008) reports that developers experience several positive experiences with scrum such as, better quality in the code, better and more frequent communication (both between the two countries and within the teams) and an improved motivation. The applied agile practices used in the project have been modified to better face the challenges faced. Some of the modification deals with challenges of being a large-scale projects, while others deal with the challenges faced in a distributed project. The practices implemented that have been modified due to the size of the project is described bellow.

- **Daily scrum meetings**, where the meetings are held in consecutive order in the same meeting room, so that it is possible for the same person to attend several meetings
- **Weekly scrum of scrums meeting** where all scrum masters and one

team member per team meets and answers the same questions they would in the daily scrum meeting. In addition, they answer two additional questions: *"Have you put any impediments in the other team's way"* and *"Do you plan to put any impediments in the other teams' way.*

- **Synchronized 4-week sprints**, which means that all the teams sprints starts and ends at the same time.
- **Separate backlogs for each team.** Each team has their own backlog, which is updated by the respective product owners.
- **Team rooms.** Each team has their own room. If someone is transferred between teams, they also change rooms.

In a case study done by Cao et al. (2004), they propose the following general guidelines on tailoring agile development methodologies to make them more suitable for development of large software systems.

- *Upfront Architecture design.* The architecture works as a backbone for the entire project. It helps to reduce the time and effort spent implementing new functionalities. It also helps reduce developers training time, and thereby mitigating the cost of bringing a new developer on board. As the systems grows in size and complexity, it is also more difficult for the developers to see the dependencies and interactions between the user stories. The architectural backbone, helps the developers maintain a certain degree of overview.
- *Short release cycles with a layered approach.* This is in many ways one of the central features of agile development, and it's still very important in large-scale and complex projects. The focus is to deliver production code that supports functionalities end-to-end, rather than developing heavily integrated modules.
- *Surrogate customer engagement.* In agile development, end-user involvement is extremely important, and in small projects, the customer is often also an end-user. The challenge when the complexity grows, the problem is amplified and the domain is often beyond the experience and expertise of a few users. The customers can therefore be surrogated by product managers or business analysts who have direct contact with the customers.

- *Flexible pair programming.* Pair programming is not seen as realistic in all situations and can be very dependent on the individual developers. In the case studied, developers paired up for analysis, design, test case development and unit testing are done in pairs. After this is done, the developers return to solo coding.

- *Identifying and managing developers.* Developers who recognise the importance of a pattern-based development and providing standard interfaces across the system are seen as a key to a successful project. Motivation and moral are also key factors. It is important that the management respect the developers contribution and not only see them as cogs in a machine. Flexible working hours, remote working, focus on results rather than micro-management, are key factors that affect the developers morale and motivation.

- *Reuse with forward refactoring.* Forward refactoring is an approach where you reuse existing code for developing new features instead of developing new solutions. Existing services remains untouched, while new services are developed based on existing ones.

- *Flatter hierarchies with controlled empowerment.* Management should be slim, and work to improve communications between stakeholders and increase productivity. Developers are empowered to make their own decisions, but exception handing is centralized to control potential damages resulting from changes that are necessary.

Experience reports

In an experience report, written by Lyon and Evans (2008), from the development of BBCs *"iplayer"*, there was one primary challenge. They used a single product backlog that was used across all teams that grew out of control. The situation was greatly improved by creating separate product backlogs for the different teams and a master product backlog which was a generalisation of the product backlog items, grouped by themes.

Another experience report by Lee (2008), describes how a large-scale project transitions from waterfall to agile through the four stages of team development proposed by Tuckman (1965). In the experience report, Lee (2008) writes that communication within the team improved drastic after the teams were gathered into team rooms, but sharing knowledge cross teams were a great challenge. Two teams could sometimes be working on a similar problem that a third team had already resolved. This was mitigated by weekly scrum of scrums meetings and daily tech leads stand-up meeting.

Articles from magazines

A more critical approach to large-scale agile development is taken by Cockburn (2000), in a magazine article,where he argues that there is one ideal method for each project size, but there's not one methodology that fits all. As the project grows larger, the less agile the methods become. The reasoning behind his statement is based upon four principles which they describes as follows.

- *A large group needs a larger methodology.* A methodology is considered larger when it contains more elements. It grows with the number of roles, and not by the number of individual people. This indicates that one should not expect small team methodology to work properly for big teams and vice versa.
- *A more critical system needs more publicly visible correctness in its construction.* This principle says greater development expenses can be justified to ensure protection from mistakes and defects.
- *A relatively small increase in methodology size or density adds a relative large amount of project cost.* Pausing development to coordinate with other people takes time and steals concentration. This does not mean that coordination is hazardous to the project progress. It does however mean that the cost increases with the number of elements in the project.
- *The most effective form of communication is interactive face-to-face, as at a whiteboard.* This principle implies that it is more efficient and therefore cost effective for developers to sit close together with easy communication. When the size of the project increases, the effective communication is reduced, and the associated costs are increased.

In an article by Boehm (2002), he argues that agile methods are difficult to scale up to larger projects due to the lack of sufficient architecture planning, but emphasises the benefits of agile development when the future requirements are highly unpredictable.

Chapter 3

Coordination

In traditional development, effective coordination is acknowledged as a critical component in organisations generally, and in software development in particular (Curtis et al. (1988)). Agile software development methodologies were particularly designed to deal with a changing environment and a high degree of uncertainty, but they de-emphasise traditional coordination mechanisms such as forward planning, extensive documentation, specific coordination roles, contracts and strict adherence to a pre-specified process. Instead they favour intensive face-to-face communication and other apparently simple practices (Strode et al. (2012)). Despite the absence of traditional coordination tools, scrum has contributed to the success of many software projects and is now being implemented in large projects, so arguably effective coordination of some type is taking place within agile software development. However, the form and nature of this coordination is not well understood.

In this chapter, the ambiguous term coordination is first described through the use of Malone and Crowson's coordination theory, in order to set a foundation for what is meant by coordination in the rest of the thesis. The next section examines the theoretical coordination mechanisms which takes place in different organisations. The subsequent chapter describes the critical path diagram, which is a coordination artefact used in the case studied. In order to better understand the issues and challenges in the aspect of coordination in large scale agile development and to put this thesis into context among other studies, the third chapter describes previous research and theories on coordination in software development, coordination in large scale projects, and a combination of the two. The final section presents a model proposed by Strode et al. (2012), as it was an important aspect of the initial motivation for the research, and has been used as a foundation for analysis.

3.1 Malone and Crowsons coordination theory

There are several different widely accepted definitions of the term coordination. Malone and Crowston (1990) defined it as follows

"When multiple actors pursue goals together, they have to do things to organise themselves that a single actor pursuing the same goals would not have to do. We call these extra organising activities coordination"

This definition was later redefined by Malone and Crowston (1994) when they redeveloped their interdisciplinary theory of coordination. **"The concept evolved into coordination being the management of dependencies."** Despite the term coordination being ambiguous, there exists a broad based theory of coordination, developed by (Malone and Crowston (1994)). Dependencies occur when a task or event's progress is halted due to the its relationship with a different task, unit or resource

Malone and Crowsons coordination theory is based on ideas from management theory, organisation, economics and computer science. The main concept is that coordination is needed to address dependencies. Dependencies arise when one action constraints a separate action in a situation. Coordination is made up of one or more coordination mechanisms, where each one addresses one or more dependencies in a situation.

While coordination theory is useful for identifying dependencies, categorising those dependencies and identifying the coordination mechanisms in a situation, it is a theory for analysis and is not intended to be used for prediction (Strode et al. (2012)). Despite the coordination theory only being a theory for analysis, it is still a valuable tool for better understanding the ways in which particular activities or artefacts support coordination in organisational settings.

3.2 Coordination mechanisms

In his synthesis of research on organisation design, Mintzberg (1980) states that organisational structuring focuses on the division of labour into a number of distinct tasks, and the coordination of all of these tasks in order to complete the labour in a unified way. Further on, he describes five different organisational structures mechanisms for coordination in an organisation.

- In *direct supervision.* One person, typically a manager, coordinates their work directly by giving specific orders to others.
- In the *standardisation of work processes.* The work is coordinated by implementing a standard guide to doing the work itself. (e.g. work orders, rules, regulations, etc)
- In the *standardisation of outputs.* The work is coordinated through standard performance measures of the outputs of the work.
- In the *standardisation of skills.* The work is coordinated through a standardisation of skills and knowledge that is usually learned before the begin to do the work.
- In *mutual adjustment.* The work is coordinated through individuals coordinating their own work through informal communication with each other.

Agile development reflects some of these mechanisms, and in particular the mutual adjustment. Mutual adjustment tends to occur in dynamic and complex environments, young organisations and organisations involved with sophisticated innovation. In extreme programming, personal horizontal coordination is achieved using pair programming and co-location. The scrum development process itself relies on scheduled and unscheduled meetings, which are achieved through sprint planning meetings, daily scrum meetings and sprint review meetings.

3.3 Critical path method

The creation of a critical path diagram, also referred to as the critical path method, is a coordination tool for planning and scheduling developed by

Kelley Jr and Walker (1959). In their article, they describe the goal of the critical path diagram as a master plan to form a basis for prediction and planning. The critical path diagram is constructed by mapping out all the activities needed to complete the project, dependencies between these and the time it takes to complete these activities. Through this diagram, one can find the longest path of planned activities, which is called the "longest path," or the "critical path." These activities are viewed as critical because a delay in them will delay the project. Additional parallel paths through the project with a shorter total time are called sub-critical or non-critical paths. The critical path diagram may also contain certain logical milestones combined with sub-critical paths in order to better distribute resources.

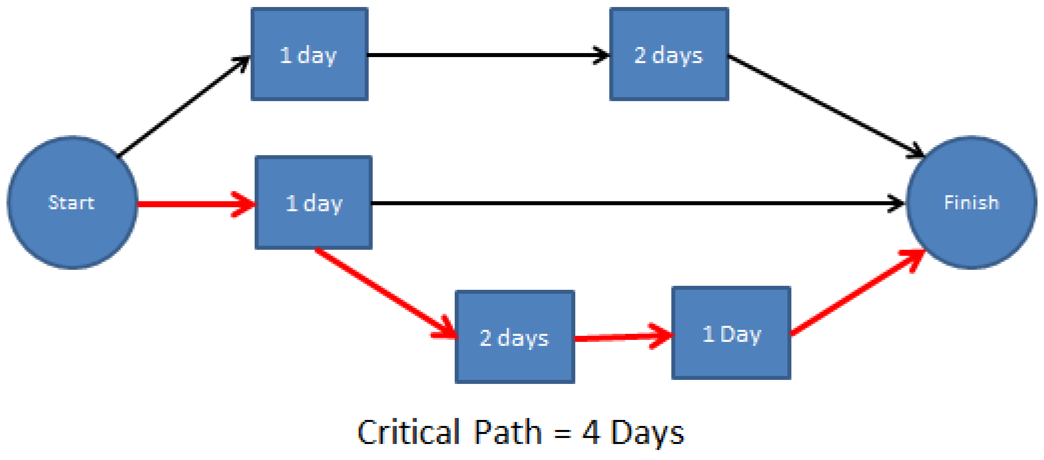


Figure 3.1: A simple example of a critical path diagram[a]

[a]http://www.acqnotes.com/Industry/Images/Critical%20Path.png

Kelley Jr and Walker (1959) describes seven benefits to the critical path diagram for planners:

1. It provides a disciplined basis for planning a project.
2. It provides a clear picture of the scope of a project that can easily be read and understood.
3. It provides a vehicle for evaluating alternative strategies and objectives.
4. It tends to prevent omission of jobs that naturally belong to the project.

5. In showing the interconnections among jobs, it pinpoints responsibilities of the various operating departements involved.

6. It is an aid to refining the design of a project.

7. It is an excellent vehicle for training project personnel.

A simple example of a critical path diagram is shown on figure 3.1. Here the blue squares indicate tasks to be done with a white text indicating how long the task will take. The arrows indicate dependencies, and the red arrows outline the critical path.

3.4 Coordination in large scale software development

This section presents relevant theory and previous research to coordination in large scale software development projects. To better understand the complexity of large scale software development, challenges and issues when scaling up software development project, this section has been divided into three sub-sections. The first subsection presents research studies related to coordination in software development. The second subsection presents organisational theory on coordination in large scale projects. The third subsection functions as a combination of the two first subsections by presenting research studies done on coordination in large scale software development projects.

3.4.1 Coordination in software development

Coordination is recognised as a critical component in software development. In this subsection, a short description of previous research done and their primary findings within the field of coordination in software development is presented.

Mishra et al. (2012) conducted an empirical study on what impact physical environment in the workspace has on communication, collaboration and coordination, in software development projects. They concluded that half-height cubicles and boards had a very positive impact on the coordination within the teams.

Hoda et al. (2010) found that despite agile teams ability to organise themselves, there was a need for a single representative for the team to coordinate between the team and the customer representatives. Sharp and Robinson (2004) identified story cards used for recording requirements, wall boards displaying story cards and their progress and unit tests as important coordination practices and artefacts in their ethnographic study of Extreme programming practices.

In a case study conducted by Pries-Heje and Pries-Heje (2011), they set out to answer "why scrum works". They found that the scrum framework provides a support for coordination, and at the same time requires very little time to foresee or negotiate the work flow. Product backlog, sprint backlog, scrum board and daily meetings were identified as four aspects of scrum which especially helped with coordination. Pries-Heje and Pries-Heje (2011) identifies coordination as one of the four critical factors to "why scrum works". Pikkarainen et al. (2008) identified sprint planning meetings, open office space and daily meetings as important practices to provide efficient communication in a case study on two co-located software projects. These practices promoted an informal communication and substituted the need for documentation as a communication mechanism. In a study by Moe et al. (2010) on teamwork in a co-located project being introduced to scrum, they found that the misapplication of scrum practices had a severe negative impact on the coordination aspect, where team members ended up not knowing what the others were doing.

3.4.2 Coordination in large-scale projects

Coordination in itself is a broad field of study. In this section, the thesis presents coordination theory related to large-scale projects. This theory is based upon large projects and large organisations in general, and not software development in particular.

Van de Ven et al. (1976) states that large-scale complex organizations and bureaucracy are often regarded as synonymous. In general, increases in size increase structural differentiation at decreasing rates, which produces a corresponding trade off between increasing complexity and cost of coordination at the administrative level, and decreasing the coordination burden within work units because activities within units tend to become more homogeneous (Van de Ven et al. (1976)). In his research, he describes two different coordination modes: Vertical and Horizontal. Vertical communication involves coordination via supervisors, while horizontal coordination occurs via one-to-one communication between individuals in a non-hierarchical relationship. His findings suggest that as the group size increases, the coordination becomes more vertical and impersonal. As complexity grows, Child (1973) states that it is likely to generate administrative problems of coordination and control. One of the ways in which such problems are dealt with, is through increased formalisation. Control is maintained in the form of standard rules, procedures and systems. In his book, Miller (1952) argues that as the size of the organisation increases, the cohesiveness decreases, and subgroup formation increases.

Van de Ven (2007) suggests that as group size grows, the face-to-face techniques of leadership behaviour gives way to more impersonal techniques of coordination. There were also an issue with conflicts within the group. The group would simply make a decision and decide how to proceed, rather than resolve the issue at hand. This is coherent with research done by Hemphill (1950), where he finds that group members are more tolerant of highly structured and directive leadership when the size of the group grows. In his research, there is also a focus on the leader role, where the demands on the leaders become increasingly more complex and numerous as the size of the group grows. Hare (1976) writes that as the size of a group grows, the member participation decreases. In line with Child (1973), Hare (1976) also draws focus towards the formalisation that occurs in larger groups, which leads to a use of more mechanical methods in order to communicate information. Thompson et al. (1967) argues that large organisations doesn't necessarily have an enormous bureaucracy and the most elaborate administration. He argues that the bureaucracy and impersonal coordination is a result of the complexity, which tends to arise in large groups and organisations.

3.4.3 Research on coordination in large-scale software development projects

In a study of large software development projects, Curtis et al. (1988) found that communication bottlenecks and breakdowns are very common. They found that a large-scale software project requires learning about the application and its environment as well as new hardware, new development tools and languages and other evolving technologies, early in the project. Software developers were required to obtain knowledge from several domains and integrate this knowledge, before they could perform their jobs accurately. In addition to integrating the knowledge of different domains themselves, there was also a constant need to share and integrate information with others. The documentation tools used were not enough to coordinate these dependencies, so informal communication arenas would emerge across team borders. They write that large projects are more successful if a single, often exceptional individual with both application-domain knowledge and software knowledge guides and coordinates the project, however this ideal is impossible for many large-scale software system.

Kraut and Streeter (1995) describes the fundamental characteristic of large software systems as being too large for one person, or even a groups ability to understand in detail. Their result suggests that personal communication is critical for successful coordination in large software development projects. Figure 3.2 displays the relationship between the extent to which software engineers used a technique to spread information and coordinate their work, and how much they valued it. The underlined techniques had values that were significantly different from what one would predict on the basis of how much they were used.
Their research suggests both formal and informal interpersonal mechanisms are used for sharing information and achieving coordination in software development projects. From fig 3.2, they also emphasise on the high value of personal communication as a tool for coordination. In their conclusion, they argue this as a potential challenge in the aspect of large-scale project as the software engineer need to acquire information and understanding from those who are remote and otherwise barred from the core of the development process as well.

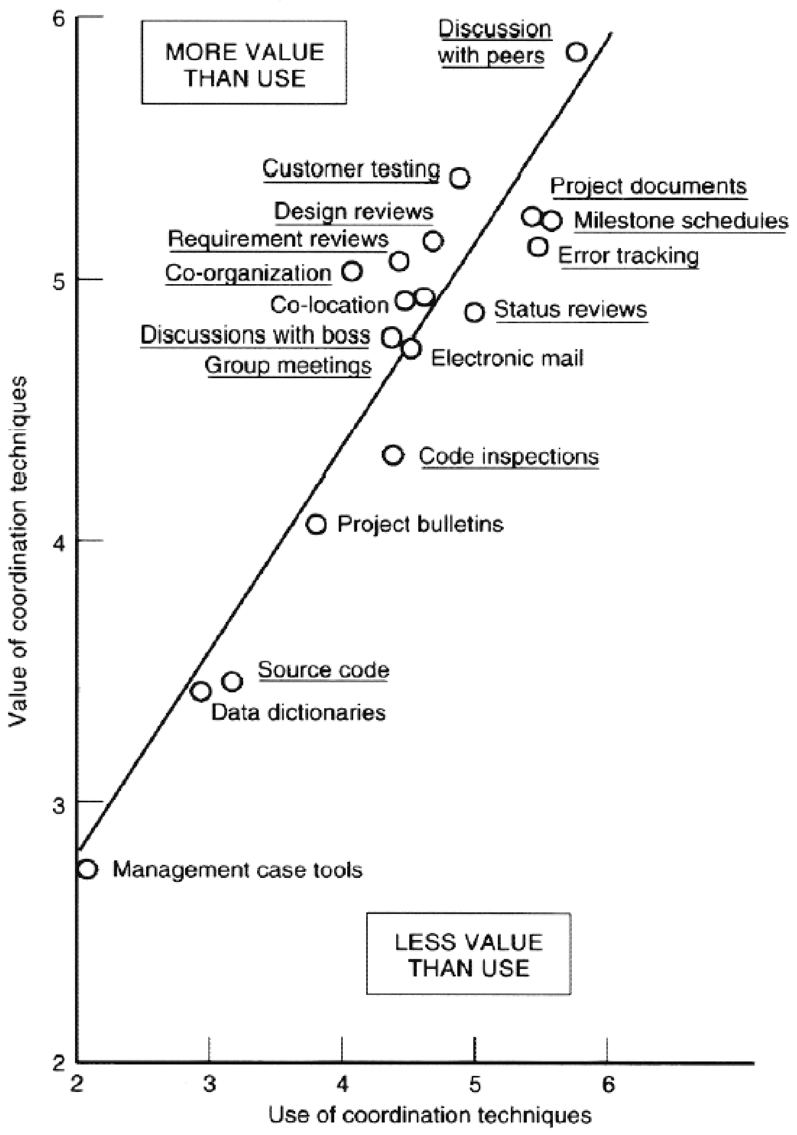


Figure 3.2: Comparison of use and value of coordination techniques (Kraut and Streeter (1995))

3.5 Strode's coordination model

In a multi-case study by Strode et al. (2012), she proposes a theoretical model for coordination in agile development projects, based upon the empirical data from her study (see figure 3.3). The cases studied were co-located agile project, and one non-agile project to contribute to contrasting evidence. Her findings show that agile development draws upon three primary component: synchronisation, structure and boundary spanning. This model has been used in this thesis as a basis for analysing the data (See section 4.6 for more information).

In the model, the coordination strategy is defined as a group of coordination mechanisms that manage dependencies in a situation. As stated above, the coordination strategy consists of three main components: Synchronisation, structure and Boundary spanning, which each has their own components. The different parts with their components is explained by Strode et al. (2012) as follows:

- **Synchronisation** is achieved with synchronisation activities and synchronisation artefacts.
  Synchronisation activities are activities that bring all of the project team members together at the same time and place for some prearranged purpose.
  Synchronisation artefacts are things produced during synchronisation activities that contain information used by all team members in accomplishing their work.

- **Structure** refers to the arrangement of, and relations between, the parts of something complex.
  Close proximity refers is a feature of agile software development methods where all project team members should be located in the same open-plan room without divisions between desks to promote an easy flow of communication.
  Availability refers to team members being available for contact with each other.
  Substitutability is when team members have expertise and skills to perform the task of another to maintain the project schedule.

- **Boundary spanning** occurs when someone within the project must interact with other organisations, or other business units, outside the

project.
Boundary spanning activity occurs when different organisations or groups within organisations, separated by location, hierarchy, or function, interact to share expertise.
Boundary spanning artefacts are physical things produced to support boundary spanning activities and enable coordination beyond the team and project boundaries.
Coordinator role is a person responsible for coordinating the communication between the team members and exterior units.

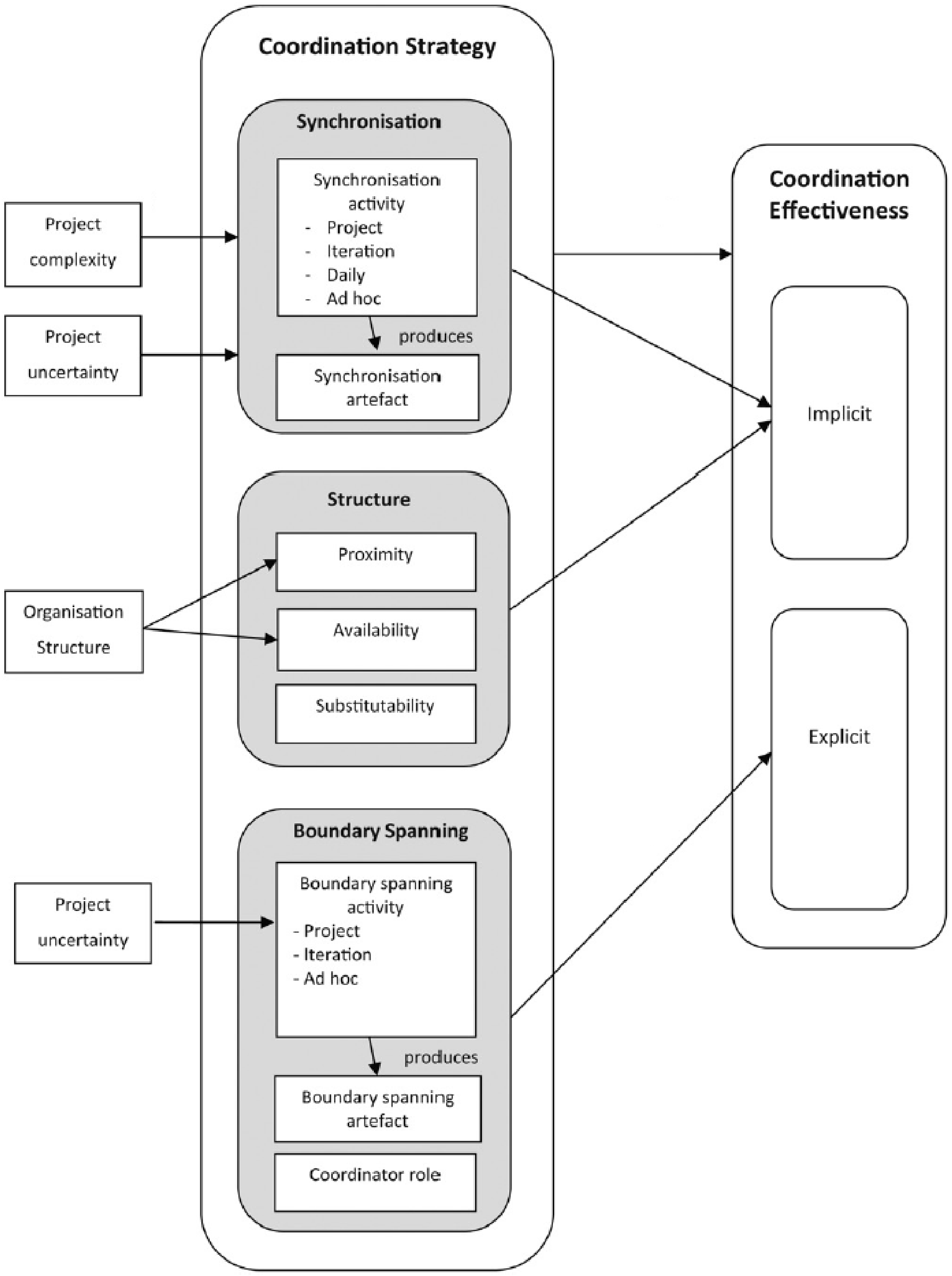


Figure 3.3: A theory of coordination in agile software development projects (Strode et al. (2012))

Chapter 4

Research design

This chapter will present how the research is designed. The first section presents the research question and the reasoning behind it, which is a vital part in designing the research. The second chapter presents the theoretical foundation on research which is used as a basis for designing this study. This is presented because in all research there exists an underlying philosophical assumption about what constitutes as valid research and which methodology is appropriate for gathering knowledge in the given study. The chosen methodology for research is imperative for the understanding gained from the research. The following section presents relevant information about the case studied and how it was conducted. Since the researcher is an important tool when conducting qualitative studies, the chapters ethical research and researchers bias, has been included to mitigate threats to validity and put the researcher in context of the study. The final chapter describes how the data was handled and processed, which is the basis of the next chapter.

4.1 Research question

In the previous chapters, there have been introduced challenges and issues regarding scaling up agile development and coordination. Since large-scale agile development is rather new, there is not a lot of literature on how effective coordination is done in these projects, or how the practises effect the coordination. The research question is therefore designed to give a better understanding and insight into these topics.

How does practices used in large-scale agile development affect coordination?

The outcome could be interesting for gaining knowledge and improving the coordination in large-scale projects. Since most the practices used in agile development originate from smaller projects, it is interesting to see the value of these when they are used in large-scale projects. The research question is not design to prove or disprove a hypothesis or theory, but rather to gain information in a field with very little literature.

4.2 Theoretical foundation

Because the theoretical foundation of research is imperative to how the research has been designed, this section contains three sub sections which gives the reader a better insight into the theoretical foundation. The first section presents a guideline to what should be included in a study, called the 6 Ps of research, which the research design has been based upon. The second section provides an overview of theoretical information about qualitative data. The final section describes the philosophical paradigms which exists within the scientific research community and which paradigm has been used for this study.

4.2.1 The 6 Ps of research

How research is done varies a lot from study to study, and there exists several views on how research should be done. One of the views is presented by Oates (2005), where she presents the 6Ps of research (see table 4.1).

Purpose	The reason for doing research
Product	The outcome of the research
Process	The sequence of activities undertaken in the project
Participants	Who you directly involve in your research
Paradigm	A pattern or model of shared way of thinking
Presentation	The means by which the research is disseminated and explained to others

Table 4.1: The 6 Ps of Research (Oates (2005))

In a parallel to the several ways to conduct research, there are also several different outcomes/products. Even given the same research question as a starting point, two different researchers could produce completely different types of knowledge and products of their research. Oates (2005) describes the following potential products: A new or improved product, a new theory, a re-interpretation of an existing theory, new or improved research tool or technique, a new improved model or perspective, an in-depth study of a particular situation, an exploration of a topic, area or field. This thesis aims primarily at producing an in-depth study of coordination in a large-scale software development project.

4.2.2 Qualitative data

There are two types of data gathered in research. Quantitative and qualitative. Quantitative is data that can be measured in terms of numbers, while qualitative data is more descriptive data which can not be measured. Gilgun (1992) defines qualitative data as data represented as words, pictures and other things not defined by numbers. Marshall (1996) states that the choice between quantitative and qualitative research should be determined by the research question, and not the preference of the researcher. This statement is on the backbone of his view upon the choice as one of the most important steps in any research project. Due to the research question, and the complex nature of coordination, this thesis used qualitative data as its data source for analysis.

Oates (2005) identifies the following as benefits when conducting qualitative research. The data and its analysis can be rich and detailed since it is not limited to that which can be measured in numbers. There is also the possibility of alternative explanation which means that different researchers may reach different, but still equally valid, conclusions. Seaman (1999) notes that the large amount of potential information a researcher can gather from qualitative research can however also be a negative aspect. There is a danger of the researcher feeling overwhelmed by the volume of qualitative data that can be gathered from just a small number of interviews. Researchers might feel swamped and unable to identify patterns and themes. In addition the analysis of qualitative data is very closely tied to the researcher. This leads to conclusions being a lot more tentative, compared to conclusion based on quantitative research. Seaman (1999) adds that when conducting qualitative research, the researcher is forced to delve into the complexity of the problem, rather than abstract it away, which leads to more informative and richer results. She also adds that qualitative data is more enjoyable for the researcher to work with, since it gives the researcher a sense of being closer to reality.

According to Galliers and Land (1987), research within the field of information systems has traditionally been biased towards traditional, empirical and quantitative research. This strong bias towards traditional, empirical and quantitative research implies that information systems is purely technological. Galliers and Land (1987) criticises this implication, and emphasises the importance of of extending the study into aspects concerning organisation and behaviour. Seaman (1999) argues that the nature of software engineering is complex due to technical issues, the awkward intersection of machine and human capabilities. Both Brooks (1975) and Curtis et al. (1988) were early to draw focus to the need for more qualitative research into both programmers and the development process. Seaman (1999) notes that the two first aspects have kept software engineering researchers engaged, and it is only recently that the focus on the last factor, human behavior, has gained significant recognition in the broader software engineering research community. The aspect of technical issues and intersection of machine and human capabilities is drastically easier to measure in a quantitative matter compared to measuring the human aspect.

Seaman (1999) credits the increased interest in qualitative data partly to practitioners, whom have seen the advances gained by adapting to research results in technical areas. She goes on to say that there are a lot of people within the software development industry who recognises that there exists a number of unique management and organisational issues, that must be addressed and solved for the field to progress.

4.2.3 The philosophical paradigm

In order to evaluate research, it is important to establish a paradigm, which is a set of shared assumptions or ways of thinking about some aspect of the world. The different philosophical paradigms have different views about the nature of our world and the way we can acquire knowledge about it (Oates (2005)). There are three primary paradigms explained in short as follows:

- **Positivism:** Our world is ordered and regular, and can be investigated objectively. Positivism is often a strong preference for quantitative data.

- **Interpretivism:** There is no single version of the truth, but different groups or cultures can see the world with different views. Researchers are not neutral. Their own assumptions, beliefs, values and actions will shape the research process. Interpretivism is often associated with gathering qualitative data.

- **Critical research:** The social reality is created and recreated by people, but social reality also possesses objective properties that tend to dominate our experience. Critical researchers criticise interpretivists for failing to analyse the pattern

In this thesis, the data is gathered with the author as the research tool. It is therefore naturally that the thesis is considered interpretive research. Klein and Myers (1999) explains interpretive research within information system as interpretive if it is assumed that our knowledge of reality is gained through social constructions such as language, consciousness, shared meanings, documents, tools and other artefacts. Interpretive research does not predefine variables, but focuses on the complexity of human sense making as the situation emerges. One of the major critics of interpretive research is the lack of a common standard evaluation criteria. The problem with finding a common standard evaluation criteria, emerges from the nature of interpretive research which does not subscribe to the idea of a set of pre-determined criteria can be applied.

4.3 Case study

This section is divided into five smaller sections with the purpose of describing how the research was conducted. The first section presents theoretical background and research on case studies as a research method. The second chapter describes how the case was chosen and the third section describes what was known about the case before gathering data. The final two sections describes how the interviews were constructed and conducted in order to gather interesting data.

4.3.1 What is a case study?

Gerring (2004) argues that for methodological purposes a case study is best defined as an in depth study of a single unit (a relatively bounded phenomenon) where the scholars aim is to elucidate features of a larger class of similar phenomena. Oates (2005) lists the characteristics of a case study as follows: A case study focuses on Depth rather than breadth, phenomenas are studied in their natural setting, it has a higher focus on the complexity of relationships and processes and how they are interconnected and interrelated rather than trying to isolate individual factors. Yin (1984) defines three types of case studies:

- *Exploratory study:* Is used to help a researcher understand a research problem or to define questions or hypotheses to be used in a subsequent study.
- *Descriptive study:* Set to describe the natural phenomena which occur within the data in question, and is often presented in a narrative form.
- *Explanatory study:* Goes further than a descriptive study, and tries to explain why events happened as they did or particular outcomes occurred.

The case study conducted in this thesis is an exploratory study, as it aims to obtain a deeper understanding about a field where there exists little literature, which may also give a better foundation for future research studies.

Some of the advantages and disadvantages with conducting case study-based research has been identified by Oates (2005).

Advantages:

- It can deal with complex situations where it is difficult to study a single factor in isolation
- It is appropriate when the researcher has little control over the events
- It is suitable for both theory building and theory testing

Disadvantages:

- It can be perceived as lacking rigour and leading to generalizations with poor credibility
- It can be difficult and time-consuming to negotiate access to the necessary settings, people and documents

It should also be noted that case studies have a large precedence within agile studies. A systematic review of empirical studies of agile software development done by Dybå and Dingsøyr (2008), shows that 72% of the studies found were from single or multi-case studies.

4.3.2 Choice of case

In order to find a large-scale project using agile development methodology, it was first defined what qualified as a large-scale agile project, before contacting companies and ask for an opportunity to study a case. In this thesis, a large-scale agile (as presented in section 2.8) is a project consisting over more than two development teams using an agile development methodology. The choice of focusing on a single case is mainly due to time constraints and the large amount of work and time related to gathering, transcribing and analysing qualitative data through interviews. Since agile development is still fairly new, and not very widespread among large-scale projects, finding a representative case invited a few challenges, but through a contact of the authors guidance supervisor from a related research project, contact was established with a company which led to the choice between two different cases that qualified for the criteria set earlier. The chosen case was selected since it was still at a development stage, and also represents a typical instance, so that the data gathered can be generalised.

4.3.3 Presentation of the case

The case is a development project of a community critical system governed by an organisation under the Norwegian ministry of petroleum and energy. As a community critical system, there exists information which is graded, and can not be published. In order to not risk publishing graded material, no pictures from the office or the people working on the project were taken.

The goal is to replace an old system that was adopted in 1993/94, and is based upon outdated technology. There is approximately 50 people working daily on the project, distributed between the customer and the supplier. The project started in autumn 2010, and through gradual deliveries, it is expected to completely replace the old system at the end of 2014/beginning of 2015. In the beginning of the project there were reports of some issues, however through a solid effort on both the customer and suppliers side, the project is now considered to be on ”the right track”, and is estimated to be completed within both the planned budget and time constraints.

The supplier is organised according to figure 4.1. The project management group consists of a project manager, a functional architect, a test manager, a technical architect and a progress manager. The team size may vary, and therefore it is not explicit that a person can only have one role. In addition to the teams, there is also one person responsible for the GUI of the product.

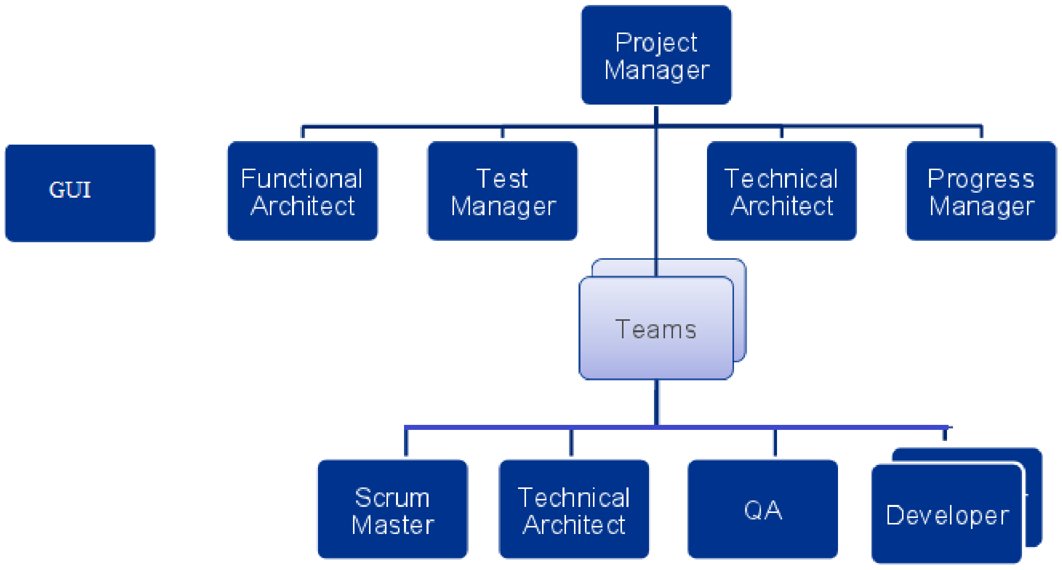


Figure 4.1: Suppliers organisation

4.3.4 Interview structure

Nandhakumar and Jones (1997) classifies interviews into three different structures, where the degree of engagement between the interviewer and the interviewee will vary based on these. The lower degree of structuring of the interview, the more the interviewer must explore the interviewee's answers. The three types are structured, semi-structured and unstructured.

- **Structured Interviews:** These use pre-determined, standardised, identical questions for every interviewee. The interview follows a pre-defined schedule with the interviewer reading out questions, and note the interviewee's responses, in often pre-coded answers (e.g. "yes, no, high, low, etc') Oates (2005) describes structured interviews as a verbal questioner where the interviewer writes down the answers directly to the computer.

- **Semi-structured interviews:** These still use a pre-determined list of themes to be covered and questions to ask, but the order of questions can be changed, depending on the flow of the conversation, and there can be other questions if the interviewee brings up issues that was not prepared in the interview guide. Semi-structured interviews can have both specific and open-ended questions which the interviewee answers according to what they think can be relevant to the theme of the question.

- **Unstructured interviews:** The interviewer introduces a topic, and then let the interviewees talk freely and develop their ideas about events, behaviour or beliefs, while the interviewer tries to not interrupt and remain as unintrusive as possible.

Before the interviews were conducted, there was little known to the author about the internal processes and structure of the project. Therefore a semi-structured interview set-up was best suited in order to explore how coordination was being performed, as well as ask questions related to theory, previous research on the field and experience reports. The interview guide was therefore constructed with more general and open-ended questions in order to obtain as much information as possible early on. This information was again used to develop new questions that could investigate further in order to obtain more information and a better understanding of the issues presented.

4.3.5 Interview guide

Interviews are recognised by Walsham (2006) as a key way of accessing the interpretations of informants in the field, and are part of most interpretive studies. Walsham (2006) also underlines the importance of keeping the interview time appropriate seeing as staff in contemporary organizations, as we all know, are normally very busy and pressured. It is essential to be sensitive to these time pressures in fixing a suitable interview time and then not overstaying ones welcome during the interview. When the author was visiting the projects office, the project was in a critical phase with a high stress-level. Despite this, they were all extremely willing to give the time needed to conduct a proper interview.

The interview guide was developed over several iterations with feedback from the guidance supervisor. Since the topic of this thesis is rather similar to the topic of Strode et al. (2012), professor Diane Strode was contacted in order to ask for permission to look into the interview guide for the paper she wrote. This both helped with improving the phrasing of the interview guide and reconsider themes that may not have been covered in the original draft. One of the most important changes that was made after reviewing professor Strodes interview guide, was to separate the interview guide into two parts. One part for project management and scrum masters, the other for developers. There are several overlapping questions, however there are certain questions that were better suited to ask developers, and others that were best suited towards management. In addition, there are some of the overlapping questions were phrased differently, depending on the role of the interviewee. The interviews were conducted in Norwegian, which means that text written in Italic in chapter 5 is not a direct quotation, but a translated quote where the meaning and intention has been preserved. The full interview guide is presented in appendix A.

4.4 Ethical research

In the studies of Walsham (2006), there are three complications, related to ethics, addressed which tends to emerge in interpretive studies. Confidentiality and anonymity, working with the organization and reporting in literature.

In order to maintain anonymity and confidentiality, all data such as personal information, project name, company name and other identifiable aspects has been kept anonymous. Any research material that can identify the people involved in the project is not released, and is being kept confidential. All the interviews were recorded with a dictaphone, which was transcribed afterwards. After the audio files were transcribed, the audio files were deleted. All the names of people, projects and companies in the transcribed material has been replaced with pseudonyms in order to safeguard the anonymity. The original transcription without pseudonyms was sent to the respective interviewees. This means that the people who were interviewed, are the only people with an original exemplar of the transcribed document.

The primary challenge of working with the organisation is usually a difference in interest of the outcome of the research. It is mitigated somewhat, although not completely, since coordination is an important factor in the organizational structure. The primary challenge is potential organisational issues that are not related to the coordination, but still relevant to the organisation that may emerge during the study.

Before the interviews were conducted, an agreement were made that the author would send the final report to the project leader when the master thesis was completed, so that the project leader could use the information gained in the final process report for the project. Walsham (2006) writes that he never agrees to reveal all his work to the organisation, which is a natural consequence of maintaining anonymity and confidentiality.

4.5 Researchers bias

In order to maintain credibility in interpretive research with the researcher as the research tool, it is imperative to be up front with anything that can affect the researchers bias. During the study, the author was a fifth year student in informatics. The author has learnt the theory behind agile development at the university, but has limited practical experience. The hands on experience with agile development was all from educational settings. The author has previously had a summer job at the company represented in the case study, and will start working there two months after the delivery of the thesis. Agile development, and scrum in particular, was very popular at the university during the study, which may contribute to a more positive bias towards agile development as a whole. Through the initial study, the author also gained a positive bias towards agile methodology, however this was strongly negated by the awareness of limited practical experience.

4.6 How the analysis was done

In the analysis, the model (see fig 3.3) introduced by Strode et al. (2012), was used as a starting point for coding the data. Through the analysis, there were themes that were represented in the data, but not in the initial coding, and initial themes that proved to not be relevant for the data material. The initial themes were therefore changed slightly to fit the themes in the data material. The description of the themes are also slightly different from the description presented by Strode et al. (2012), in order to better fit the coordination challenges in relation to a large-scale project. The result was four themes. An overview of the themes with sub-themes are described below. The results of the analysis is presented in chapter 5.

4.6.1 Synchronisation

Synchronisation refers to activities and artefacts that helps the teams achieve synchronisation in their work.

- *Synchronisation activities* are activities that bring together people from different teams at a time and place for a pre-arranged purpose, where the goal is to improve a common understanding of the task, process, the expertise of other team members, or to work together against a common goal.

- *Synchronisation artefact* refers to artefacts that helps to synchronise the project. They can either be a result of synchronisation activities, or tools implemented by the project management in order improve the communication/coordination/progress.

4.6.2 Structure

Structure refers to the relationship between parts of something complex.

- *Proximity* refers to the physical closeness of members in the project. In agile development, close proximity is a feature. All members of the project should be located in the same open plan room without divisions between the desks.
- *Substitutability* refers to the ability to substitute a person, while still keeping the time schedule.

4.6.3 Boundary spanning

Boundary spanning occurs when someone within the project must interact with other organisations or someone outside the project to achieve project goals. This project could be seen as a large project which combines both the customers work and the suppliers work into one big project, but since they are two separate organisations, I have chosen to limit the project to the supplier side, and therefore treat the customers as an external organisation.

- *Boundary spanning activities* are activities performed, where interaction with other units or organisations is needed.
- *Coordination role* is a role taken by a project member, with goal to better improve the coordination, and better support the interaction with people who are not on the project.

4.6.4 Reactive coordination

Reactive coordination, is coordination activities that happens as a reaction to an event.

- *Individual responsibility* refers the responsibility of each team member to coordinate activities when it is needed.

Chapter 5

Results

In this chapter, the result of the analysis will be presented. The subsequent sections consists of the four themes synchronisation, structure, boundary spanning and reactive coordination, with content according to the description of the themes found in section 4.6. Whenever there is a difference in opinion between developers, scrum masters or project management, has also been presented in this chapter. However, there were no clear systematic coherence between opinions and role in the project.

5.1 Synchronisation

There are several synchronisation activities and artefacts used in the project. In the following sub-sections, the results of the data analysis is presented. A description of the themes can be found in section 4.6.1

5.1.1 Synchronisation activity

Project synchronisation activities occur in several different formats in the project. In the beginning of the project they used pair programming due to a very positive experience on a previous large-scale agile project. The effect of the pair programming in this case was however described as both a positive and a negative experience. Those in the project management group who had been there when pair programming was used said that it had not given the effect they had hoped for in this project, while the developers who had been part of it said it was very helpful, but after a while they simply didn't have time to it because of time constraints.

"We've used a bit of pair programming. Especially in the beginning of the project, and less now near the end. In the beginning it was very good to share experiences and knowledge. Not only about the domain, but also on how you should write code."

One activity that the lead technical architect pointed out as an important coordination activity that they felt yielded good results in both planning and organising the partial deliveries, was to create a Critical Path Diagram.

"Last year we started making dependency diagrams between the user stories, similar to PERT diagram or Critical Path Method, in order to find a critical line, and find what epics had the most risk. In addition we did it so that we could see what areas was possible to develop in parallel between different smaller teams, or potentially what dependency we had to process in order to make it possible"

None of the developers pointed towards the critical path diagram in explicitly, but several of them pointed out that the tasks were distributed among the teams so that there were nearly none cross-team dependencies. It was only recently that a part of the system was too big for one team to develop and a separation without dependencies wasn't possible.

The architects have what they call an architect forum, which is a forum where they can discuss different technical problems and solutions. It also works as a forum to discuss dependencies between teams. This is mentioned by all the interviewees with an architect role, as an important tool for coordination.

"The architects meets to discuss technical solutions and dependencies between the teams."

The scrum masters also meet in similar fashion in a scrum of scrums meeting. The scrum of scrums meetings are held once every week and are open to all, including the costumer, but it is usually only the scrum masters and the contracts manager in the project management team who attends. Scrum of scrum meetings are used to share information, and talk about issues that might have occurred, since there can often be similar issues. At an earlier stage, when some teams were working better than others, scrum of scrum was also used to try to figure out what factors made a team work better than another and what could be done to improve the other team. The challenge with these meetings, is that it is impossible for the scrum master to keep an overview of all the small details of all the team members and it has been at the detail level where the need for clear communication has been a necessity.

"Scrum of scrum tends to work rather well. It depends on how much you want to share and how much you can remember of small details, because it tends to be on a small detail level that you need a clear communication."

There is also a discrepancy between the scrum masters and the developers when it comes to who is responsible for uncovering unforeseen dependencies between teams. A few of the developers feel that the scrum masters should uncover it through scrum of scrum meetings, while scrum masters say that the developers has an individual responsibility to make sure that what they're doing doesn't block the work of others. The individual responsibility will be presented further in section 5.4.1.

"If you find that two people are working on the same thing, we tend to move the communication between the two respective team members."

Through the initial planning and the different communication arenas, good synchronisation and thereby coordination is achieved, however there has still been issues. There has been situations where teams work on very similar things, but there is low or no cooperation and therefore they may end up with two completely different things.

"Two teams can work on a very similar thing and still do it completely different"

Stand up meeting is a standard part of the scrum method and was also used in this project. As a response to two different teams having user stories where there were cross-team dependencies, it was decided to keep the morning standup meeting at different times, so that members from one team could attend the other teams.

"A standup meeting in the morning helps us make the project more efficient."

Most interviewees mentioned standup meetings when talking about what practices they felt were useful in this project. Nobody expressed a negative attitude towards the standup meetings.

Retrospective meeting is another tool from scrum that has helped increase the synchronisation and is being held every third week. They are however primarily aimed towards enhancing the coordination and maintain a continuous improvement within the teams. In addition to the coordination and continuous improvement aspect, it is also a good tool for people who are new in their role to get a better understanding of how they themselves can improve, which again lowers the threshold for changing roles within the project. Therefore the retrospective meetings improves both the synchronisation within the teams, as well as improve the substitutability.

”Retrospective has worked very well. It was particularly useful for me, since much of it was rather new, to get feedback from my team about what and how we wanted to do things and reflect on what we can do to make things work even better.”

”Retrospective meetings are a lot more important than how they are presented in theory.”

There are also meetings between the functional architects and the people responsible for testing on the different teams, but these meetings operate on a more ”when needed” level.

5.1.2 Synchronisation artefact

In the studied project, the synchronisation artefacts include Kanban whiteboard where the progress of the user stories are presented, the user stories themselves, reports from the leader group, dependency diagrams, an hindrance log where developers can add things that hinders them from working as efficient as possible and work items. There were also invisible artefacts such as JIRA, the architecture, code standards, source code control and unit tests. Kanban was mentioned explicitly by some of the developers as a useful tool for coordination because it helped with visualising tasks to be done and created transparency.

"One of the things that may have helped us the most is visualising things by putting things on the board."

5.2 Structure

This section describes how proximity and substitutability impacted coordination in the project. A further description of the themes can be found in section 4.6.2.

5.2.1 Proximity

The project was set up in a large open area, where groups of people were separated by either whiteboards, or a low wall. All interviewees agreed that close proximity is a must for good communication.

"You have the formal arenas like meetings, mails etc, but the informal discussion you get at the coffee machine in the morning when you get here is priceless."

When asked about the open office landscape, all interviewees raised two factors. Increased communication and noise level. These two were weighted up against each other, but all interviewees ended up with the positive outweighing the negative aspect. There were three groups that especially enjoyed the open office structure. The project management group, the developers new to the project and the and inexperienced developers. The project management group said they tried to keep an overall view on what was going on in the project and liked to hear what was being discussed. The newer members of the project and the more inexperienced developers particularly enjoyed the open office structure because it was very easy to initiate contact with other members of the project

"I primarily think it's very nice. It is very easy to contact others, and that leads to a very low threshold asking for help, which I do rather often."

A higher level of communication was a clear positive attribute to the close proximity factor, however the need for communication was mainly within the teams. A few of the developers said that they sometimes could pick up interesting things from overhearing what other people said, but usually the communication need was within their own team. Three of the developers noted that they would have preferred a larger separation of the teams on the foundation of what information they actually needed, and where they felt communication was most important. They believed this would still maintain the important communication within the team, while at the same time lowering the general everyday noise.

"I believe separate team rooms would be the ideal, because it is the communication between team members that has the highest value. What the other teams talk about can sometimes be interesting, but there is a high chance that there is nothing relevant for you there, which leads to the ratio between information and noise is lower than if you had team rooms. Of course, you loose some information between the teams, but one of the benefits of us all working for the same company, is that people start to know each other after some time, and you still have the coffee machine and the fruit basket if you need to ramble a bit"

In order to negate the negative effect of noise, almost all members of the project had headphones on their desk, which they would use to listen to music while working. All the interviewees except one said they felt like they were able to concentrate and work on what they needed despite the noise.

One of the interviewees also mentioned that he encountered a problem with the close proximity when he needed to work isolated and creatively.

"In periods where I need to work more creatively and isolated, I tend to use home office. I find it a lot easier to work from home than to use a quite room at work, because then you are still available."

5.2.2 Substitutability

The results for substitutability varies from the different interviewees. The management group has a strategy that anyone can be hit by the bus tomorrow, and therefore, nobody can be indispensable. At the same time, they do point out that as the code base and complexity grows, it is nearly impossible for everyone to know everything, and a certain level of specialisation has emerged.

"In the beginning, when everyone knew everything, then almost everyone could fix any problem. Now the solution is enormous, and it is impossible for everyone to know everything. So we do have a certain degree of specialisation, but not an area that only one person knows."

One developer is very critical to the way the emerging specialisation is handled by the project management group. He feels that when there are tasks that needs to be done, the project management goes directly to certain people to ask if they can do the task, rather than give the task to the team, and let them distribute the tasks among themselves. He feels that this creates a degree of favouritism, and washes away the team concept.

In order to achieve substitutability, there has been some role rotation, which creates redundant competence and decrease the negative effect it would have on the project if someone had to be substituted. The pair programming also helped improve the substitutability of the project.

Despite the project managements effort to achieve substitutability, several of the developers, and especially those that are newer to the project, still feels there are people on the project that are essential. They say the project wouldn't stop if they disappeared, but it would definitely hurt the project progress.

"I do notice when the right people are working from home, because then all of a sudden there are certain things that are a lot more difficult to get done."

When asked what makes certain people essential, it is not the technical competence that is valued as essential, but the tacit knowledge achieved by simply being part of the project for a long time. The people who had been part of the project the longest, were also the ones that were pointed out as people with key knowledge. Through being part of the project for a long time, these people had gained tacit knowledge and a general better overview of the project as a whole.

"There is absolutely people with key knowledge. It's the ones who have cared the most, worked the hardest, been here the longest, and taken ownership to

part of the development, code or functionality."

"There are not so many people who have been here from the beginning, but fortunately, there are more than one. I think that we who've been here from the beginning have some tacit knowledge that is difficult to share with others."

Abseloutly everyone who said there was people on the project with key knowledge, mentioned the lead technical architect as a person with key knowledge. This was generally because he had been here since the beginning, and he had some part in nearly everything that had been done in the project. The lead technical architect himself was aware that a lot of people would probably point at him, but argued that this was not the case. Earlier in the project, he might have had key knowledge, but as the project has grown, the combination of what others possess, covers the same knowledge that he possesses.

"A lot of people will probably point at me, but I firmly believe that we have covered my competence, though it might be a bit spread. I'm a bit of the old man in the house here. I've been here the longest. I've had a central role for the last three years, and I know most parts of the system, but I don't believe that any of my competence is irreplaceable."

5.3 Boundary spanning

In this project, most communication and interaction outside the project was towards the customer. There were also a few interactions with smaller projects, but these didn't bring any noteworthy coordination or organisational challenges to the progress of the project. There were also stakeholders and communication between the customer and other external parties that the customer had to deal with, but this exceeds the scope of this report. This section presents the results of the data analysis related to proximity and substitutability. A further description of the themes can be found in section 4.6.3.

5.3.1 Boundary spanning activity

In the beginning fo the project, there was a skewed distribution between the suppliers need for manpower on the customers side, and the actual manpower available. This turned into a bottleneck, because the customer simply didn't have the capacity to support the needs of the supplier.

"In the beginning the customer didn't have the capacity to support the amount of specifications and questions as we would have wanted."

This uneven distribution naturally caused some turbulence for the project in the first period. It was however clarified in a renegotiation of the contract, and after that the customer scaled up their manpower and capacity. All Interviewees who were present before and after the renegotiation points out the renegotiation as an extremely important activity for the success and progress of the project.

"In the renegotiation of the contract, it was made clear that the customer simply had too little manpower. They have staffed up quite a lot, which we definitely notice on the flow of the project."

Another boundary spanning activity that has been improved later was the construction of user stories. Developers report that in the early stages of the project, the description were often too small and not specified enough. Developers and scrum masters report of varied improvement rate on the specifications of user stories where some report they almost give too many details of specifications, while others still feel the specifications are too thin.

"There were some challenges in the description of the solution in this delivery. It was not completely finished, so I felt we started at the wrong end."

Despite the increase in manpower, clarification with the customer is still a bottleneck. This is not due to the customer not scaling up enough, but there is certain knowledge and competence that is shared among a few people. In addition, the proprietors of this information tends to be very busy.

"Key competence on the customer side has been with only a few people who has not been involved in a large degree with the project."

This leads to certain questions taking a long time to clarify. Especially if the decision affects other branches of the customers organisation or other stakeholder. The long turnaround time has been a problem, and something a central person in the project management group points out as something he wished had been done better.

"If I had to do things over, I would set a demand to turnaround time on decisions."

When talking about dependencies, the developers also pointed out the time they wait for a response from the customer as a major factor. When bugs or errors are found, it is often a question if it is a discrepancy or a change, which one developer described as a question about who has to pay for it.

"Clarification from the customer is definitely what we spend the most time waiting for."

Despite this being a point of frustration at times, the developers felt they had a good relationship with the customer, and the threshold for going over to talk to them or ask them questions was very low. When asked why this threshold was so low, the developers pointed to three factors. The project had been going on for such a long time that they knew most of the people on the customer side, the difficult business discussions were raised and moved away from the project and the fact that the customer worked in the other wing on the same floor of the building. Two of the developers said that they would have preferred to have the customer even closer, the small distance still raised the threshold a little bit, but their current setting still worked rather well. One developer agreed that close proximity was an important factor, but that in an agile project, you need to be close to the customer constantly, and although it had improved, it was still not good enough simply because the customer didn't have the time nor the resources to be available constantly.

"I believe that a critical success factor in any project is that the customer and supplier are collocated."

Both the collocation and the the length of the project were viewed as a strictly positive effect on the communication between the supplier and the costumer. The abstraction of the contract discussions out of the project had different views among the developers. Two of the developers mentioned this as a critical factor in the good interaction between the developers and the customer.

"It is important that we have moved the difficult commercial discussions away from the project."

The negative aspect of this was brought up by a developer who felt that some of these discussions were too important and affected his work to much for him to not know what was going on. He was not directly negative to having the discussions moved away from from the developers, but the information about what was being discussed, and what had been decided was inadequate. The developer felt that this information was extremely important, since it was often information that would affect him or the whole project somewhere down the line, that it was not sufficient to present them in bits and pieces. He also stated that he understood that not everything could be presented constantly, because that would create too much overhead for the project, but the information should be made available so that those who are interested can extract the information they find relevant.

"Some time back we had a couple of clean up meetings. After that we got a few sentences in bits and pieces about what's going on, and I feel that is not enough. It affects us too much. They try to say that it shouldn't affect us, but it just doesn't work that way."

Another activity that the project management group find highly valuable, is the customer sending over testers to sit in the team.

"The customer has sent their available resources to sit and test with us, and that is worth it's weight in gold, because in many cases they can see discrepancies long before we can."

This is also emphasised by a developer who believes this was a key factor to the success of a previous large-scale agile project he worked on, although he would not only like to have the customers sit there and test, but have users directly involved in the project. In his previous project, their team had a user within their team, which means he had first hand experience with the system they were replacing. This is something he feels should be strived for in a project.

5.3.2 Coordination role

In each team there is a person that acts as responsible for the communication between their team and the costumer, which is usually the scrum master. The scrum masters interviewed said that when team members need information from the costumer, they should first check with the them to see if they already have the information within the project.

"I think it's important to have a few people who are responsible for the communication with the customer. Especially when we're as many as we are."

This role is explained as very important in such a large project, because when things get complicated and a lot of questions arise, there can't be several different people interacting with the costumer, asking questions that they may already have answered. In addition, similar questions can be asked to different people on the costumer side, who may give different answers, which is not good.

One scrum master also differentiates between technical and functional questions, and says that functional questions should be ran by her first, but if it's technical questions, the developer can go directly to the costumer. Regardless if it's a technical or functional question, she says that she would like to be updated and kept in the copy field in emails, so that she can maintain a general overview.

"If it is functional questions, he should ask me first, because it might be that I already have the answer."

5.4 Reactive coordination

Identifying and reacting to unknown dependencies was an important aspect of coordination in the project. This section presents the results from the data analysis related to individual responsibility. A further description can be found in section 4.6.4.

5.4.1 Individual responsibility

Due to the way the user stories were divided, there were, up until recently, not many cross team dependencies. In the latest partial delivery however, it was not possible to divide the stories up in such a manner. When the developers were asked what they did to avoid affecting the work others have done or are doing, the developers pointed towards the tests, and said that it was peoples own responsibility to write good enough tests for themselves in order to assure that other peoples work doesn't effect them.

"To a certain degree, you have to trust that people have written and tested it well enough. So if everything is green, then things should be okay. Especially if you are fixing code that someone else has written, because then you don't know all the little tweaks they might have done."

"I think it is every teams responsibility to write tests that makes it difficult to ruin anything unless you have explicit intentions to do so."

The developers also said that the other developers were good at writing sufficient tests, and they trusted that the tests were good enough, so that they did not risk accidentally interfering with the work that others had done. The only issue was with GUI, which some developers claimed were either too complicated to write sufficient tests, or they simply didn't have the competence or experience to write sufficient tests. This was somewhat mitigated by bringing up the GUI, and checking that there were not any immediate problems. When cross team dependencies were discovered, it was also the individual developers in the different teams responsibility to create and maintain communication. Some used pair programming, while others preferred to work separate. The choice was up to the developers themselves, depending on their preference.

There are also several interviewees who point to the importance of peoples mentality as a coordination mechanism. They say that it is essential that all individuals on a project are interested in continuously improving the process, and takes responsibility to see the improvements done. Through this drive for continuous improvement, coordination needs are met and the process itself evolves.

"A lot of people here are concerned with always doing things better, and that makes it so that things are better coordinated as time goes by."

Chapter 6

Discussion

The objective in this thesis was to examine the research question through an exploratory case study. The study gave the author a total of 17 interviews to decompose and analyse, which is presented in chapter 5. This chapter contains a discussion of the results in order to provide an answer to the research question.*"how does practices used in large-scale agile development affect coordination."* This is done by first discussing the chosen model for scaling up agile development up against previous research, which gives a better understanding of how the practices are implemented and how it compares to existing guidelines. Second, this chapter will look into the coordination mechanisms and factors in order to better understand the underlying theoretical coordination foundation within the project. After that, the thesis will look at the how specific practices works in the project and what impact they have on coordination. The fourth section describes the limitations and implications of the study, in order to better understand what context factors exists, so that the reader can better evaluate if the findings of this study is applicable on other similar projects and further discuss what implications these findings have on existing theory and guidelines. This chapter also contains an evaluation of the study, which describes challenges faced during the study and brings up issues concerning trustworthiness and generalisation of the case. Finally, a quick summary of the discussion chapter is presented.

6.1 Comparison to guidelines for scaling up agile development.

How the project has scaled up agile development to fit a large-scale project is not one pre-defined by existing guidelines, but rather an continuous evolutionary process. As the project has grown, it has become nearly impossible for one person to keep a complete overview of the entire code base and a certain degree of speciality has emerged. The emerging speciality in domain knowledge and competence is a step away from the guidelines that Larman and Vodde (2008) presents, where all teams should be able to complete any item in the backlog.

This leads the project much closer to Ecksteins (2004) strategy for large-scale agile development, through the use of feature teams. The results show that this division of labour works well when development goes according to plan, but creates a challenge when a team is finished earlier than scheduled. This is because it makes it very difficult for a team that finishes early to relieve the workload of different team. One could therefore argue that the introduction of a critical path diagram works as a coordination mechanism by reducing the dependencies and need for coordination between the teams, but it comes at the cost of reduced agility and flexibility. Although the emerging use of feature teams is a step away from Larman and Voddes (2008) guidelines, the project still uses several of the practices introduced by Larman and Vodde (2008), such as scrum of scrums meetings and joint retrospective meetings. It is also evident that the project has grown in size and complexity beyond the ability of one person to understand in detail, which is one of the fundamental characteristics of large software systems according to Kraut and Streeter (1995).

In comparison to the seven guidelines Cao et al. (2004) on how to tailor agile methodologies to make them more suitable for large scale development, there are some similarities, but also some defects. The architecture in the project is an important foundation and functions as a backbone for the entire project, but it is not a strictly upfront architecture. Surrogated customer engagement and reuse with forward factoring are not present in the project and pair programming is used in a more ad hoc fashion, compared the guidelines described by Cao et al. (2004). The biggest similarities with this project and the guidelines are short release cycles with a layered approach and a flat hierarchy with controlled empowerment. Motivated developers is also present in the project, but they are not recruited based on their recognition of importance of pattern-based development, as this is not seen as a key to success in this project.

6.2 Coordination mechanisms and factors

This section will look at the mechanisms introduced by Mintzberg (1980) and how they exists within the project. There are also other factors that have had an impact on how coordination is done in the project, which is described in the second and third sub section.

6.2.1 Mintzbergs coordination mechanisms

Among the five coordination mechanisms introduced by Mintzberg (1980), it is mutual adjustment, direct supervision and to some degree standardisation of work process, that are the most prominent mechanisms in the project. The mutual adjustment is referred to as priceless by the project management and exists both within the the teams and across the teams. Mutual adjustment is in many ways a foundation in agile development, unlike direct supervision which is rather discouraged. The direct supervision can be found in the project through the distribution of absolute deadlines, the construction of a critical path diagram and management personally distributing tasks to people with special competence or domain knowledge. It is unclear whether the use of direct supervision is a result of the project size or the complexity of the project. When it comes to the distribution of tasks to people with special competence or domain knowledge as a coordination mechanism, it is unclear if this is an evolution based on necessity, or convenience for the project management. A proposed solution by one developer was to distribute the task to the teams and let team decide how to deal with it.

In terms of standardisation of work process as a coordination mechanism, one could argue that defining and using a methodology in itself is a coordination mechanism. The results show that there are several coordination arenas e.g. architect forum and scrum of scrum meetings, that are planned structured meetings created to enhance the level of coordination. These are all a result of a chosen methodology, which would suggest that standardisation of work process is also an important coordination mechanism in the project.

6.2.2 The value of experience

The ideal situation for coordination in a large-scale software development project is described by Curtis et al. (1988) as a situation where you have one exceptional individual with both application domain knowledge and software knowledge which guides and coordinates the project. There is not one such individual in the project, but it is apparent that tacit knowledge is looked upon as both valuable and important. The size of the project doesn't only refer to the amount of people working on it, but also the large timespan of the project. When asked if there are people on the project with key competence or key knowledge, it was always people with key knowledge that was mentioned. In addition the knowledge people viewed upon as important was the tacit knowledge obtained through being part of the project for such a long time. This tacit knowledge combined with a coordination role, helps improve the coordination of the project and is valued greatly. This would suggest that a low turnaround of staff, especially on management level, would be a great benefit for a large-scale project.

6.2.3 Boundary spanning

Although the project uses Jira to track and manage issues and dependencies, the developers say that they prefer to simply walk over to the customer and talk to them when they need clarification or information. Even though this is the preferred way of communicating, they also use Jira for documentation after they have clarified something with the costumer. This implies that the developers value the personal communication, which is in line with the findings of Kraut and Streeter (1995) and also contributes to maintaining a horizontal coordination mode (Van de Ven et al. (1976)).

Although the communication works well, there is still room for improvement when it comes to coordinating boundary spanning dependencies. Developers report of a long feedback loop or time on deviations and change orders. There is also issues regarding the specifications which leads to developers not being able to work, and have to spend time seeking out information and clarifications which again may take a long time. This is an area mentioned explicitly by several among both management and developers as an area with room for improvement. Waiting on clarification from the customer is also mentioned most frequently by the developers when talking about what dependencies they spend the most time waiting on.

6.3 Practices

In order to better answer the research question, this section presents a discussion of how the different practices impact coordination within the project, the practices are also discussed up against relevant research presented in chapter 2 and 3.

Test driven development

The primary benefit with using test driven development is described by Erdogmus et al. (2005) as an increased quality insurance in the code written. The data gathered in this study does not confirm, nor deny this hypothesis, as quality of code is not analysed, however there are other benefits of TDD displayed in the case. The developers describe an almost blind faith in tests written by others. As long as others have written good enough tests, the developers don't risk interfering with or breaking functional code. Some developers and architects said there was room for improvement when refactoring code that may impact the work of others through e.g. sending out an email to inform the others, but the responsibility was primarily on the other person to write sufficient tests. One downside is on the GUI side. It is mentioned in the results that tests on the GUI side are not always very good and may be due to the complexity of automated GUI tests, or the lack of knowledge of writing GUI test code. This is mitigated by bringing up the GUI and checking that the new code doesn't haven't changed anything notably. This suggests that test driven development has a very positive impact on coordination both internally in teams and across teams, as it helps to discover dependencies and avoid conflicting code.

Critical path method

Using a critical path diagram as the basis for further planning was only mentioned as a valuable tool by the project management, where it is described as a tool that yields good results in terms of coordination. Even though it is not mentioned explicit by the developers, the importance is still reflected in their answers by reflection of the low amount of cross-team dependencies that exist within the project. This low amount of cross team dependencies is an important difference from previous experience reports (Lee (2008), Lyon and Evans (2008)). Through the use of critical path diagram, the management is able to divide and distribute work in the teams product backlog, so that the need for coordination across the teams is greatly reduced. It is also worth mentioning that critical path diagram is not an agile practice, and can be said to reduce the agility of the project since it introduces substantial planning to the project.

Although the critical path diagram has a positive effect on the internal coordination, it is not without side effects. The knowledge of a certain domain is solely within the team that develops a certain feature. This is not a challenge to substitutability, as there are others within the teams with domain expertise, but it makes it difficult for a person on team B to develop an item that is intended for team A.

The critical path method has yielded good results in the project, but it does not help the project when bundles of tasks that are dependent of each other become too big for one team to handle. One of the interviewed scrum masters reported that the previous partial delivery was simply too big to be handled by one team. This suggests that although the critical path diagrams can yield great results in a large size project. It is a tool that reduces the need for coordination across teams, but in this project there are no signs of it handling the dependencies across teams that arise when a complete separation is not possible. This means that although it is a good tool for coordination, it still relies heavily on other coordination mechanisms in order to function optimally.

Co-location

Co-location is a practice recommended in both Scrum (Schwaber (1997)) and extreme programming (Beck and Andres (2004)), but it is not always possible e.g. in distributed projects on a global level. On this project however, it seems to be an imperative factor for success of the project. The ability to communicate and coordinate within the teams are clearly important, but it is unclear how efficient the open landscape is for cross-team coordination, as the information within other teams is seldom valuable for the developers. The interviewees reports of a low or non-existing threshold for contacting other people on the project, which helps to maintain a high degree of vertical communication and makes mutual adjustment across teams much easier, but it comes at the cost of a higher noise level. Since the critical path diagram helps to reduce the need for cross-team coordination, the yield in terms of coordination compared to the noise level is very low. The noise level is so disturbing that most people on the project choose to use headphones, which may reduce communication and the coordination within the teams in a negative manner. On this foundation, it would seem as if team rooms as used by Lee (2008), are a better solution for large-scale agile development.

Scrum of scrums

Scrum of scrums is implemented in the same manner as described by Larman and Vodde (2008). The scrum of scrums meetings are held once every week and is open for all. Although it is open for all, it is usually only the scrum masters and the contracts manager in the project management team that attends these meetings. Scrum of scrums helps distribute and coordinate known dependencies across teams, but struggles to identify unknown coordination needs. One developer mentions that it is on a more detailed level, there really is a need for coordination. Since it's close to impossible for a scrum master to keep a detailed overview of everything happening within the team, it also makes it difficult for scrum masters to detect these needs on the scrum of scrums meeting.

Architect forum

The architects mentions the architect forum as an important forum for the architects to discuss and coordinate work related to the software architecture, resulting in a well defined architecture, which is the foundation which the developers build upon. This is supported by Cao et al. (2004) where they describe a good architecture design as a key factor for a successful large-scale agile project.

Standup meeting

The standup meetings are slightly different from existing guidelines on large-scale agile development (Larman and Vodde (2008), Eckstein (2004)), where each team holds separate standup meetings in order to keep updated on the work being done within the team. The teams still hold separate standup meetings, but it is explicitly not done in parallel. In order to allow for better coordination, the standup meetings are held on different times so that developers on other teams can attend. This is a practice mentioned by several interviewees as a good practice used in the project, but developers mentions that there is still room for improvement in disseminating the information, about what's happening in the other team, to the rest of the team. This arrangement of having standup meetings in different times has evolved on the background of needs, and is something also found in the case studied by Paasivaara et al. (2008). Daily standup meetings is an important practice to improve coordination and provide efficient coordination, which is coherent with the findings of Pikkarainen et al. (2008) and Pries-Heje and Pries-Heje (2011). Pikkarainen et al. (2008) suggests that daily standup meetings promote an informal communication and substitutes the need for documentation as a communication mechanism. The low degree of documentation as coordination tools used in this project supports their findings.

Kanban

Kanban is used in the project, and is valued by the developers as tool for visualisation. This visualisation is an important tool for coordination within the team and creates transparency for other teams when there are cross-team dependencies. This transparency is one of the key features of kanban according to Cocco et al. (2011). It is through this transparency that the developers are able to coordinate known dependencies, as they can see where tasks that has dependencies connected to them, are in the development process.

Retrospective meetings

In this project, retrospective meetings has been an important tool for continuously improving the process, which some contributes as the key factor to the success of the project. Both the developers and management points to the retrospective meetings as a critical tool to continuously improving the process. It is also noted that for retrospective meetings to work, the project is depending on people who are not just interested in doing their own work, but also interested in improving the process.

6.4 Limitations and implications

This section will first look at the limitations of the study, which presents factors that have impacted how the project works. This is done so that the reader will better understand the context of the project, and gain a better foundation for evaluating if the findings of this study is applicable to similar projects. The next subsection will focus on what implications this study have on existing theory and guidelines. The final subsection will take a look at what this study means for the target audience of the report.

6.4.1 Limitations

There are several factors which are important to describe in order to fully understand the context of the study and to what degree the study is applicable to other projects. One major factor that contributes to the success of the project and the success of the practices used is the fact that the system being developed is designed to replace an old system. Although not all dependencies are discovered in the planning stage, it makes it easier to identify dependencies earlier in the process and improves the accuracy of the requirement specifications. This may be an important factor in the success of the critical path method. It is therefore uncertain whether this is a practice that will yield good results in a large scale project that is being developed from the ground up. Another important factor is that all the people working on the supplier side is from the same consultant company, which is contributed as an important factor to maintaining a very low threshold for communicating with each other. Although co-location which is standard in agile development, it is still an important factor that must be taken into account when evaluating if the practices used in this project can be applicable in other projects. The combination of everyone working for the same company and co-location is important for the mutual adjustment coordination taking place in the project. There is a low degree of synchronisation artefacts, which suggests that developers and management prefer direct contact.

6.4.2 Implications for existing theory and guidelines

The methodology used in this project has evolved over time- It has evolved further away from scrum, which implies that simply scaling up the scrum practices as by Larman and Vodde (2008) suggests, may not be enough to handle the complexity of large-scale software development. The increased specialisation also suggests that feature teams, as described by Eckstein (2004), functions well in large scale agile development projects. Erdogmus et al. (2005) describes two positive properties of test driven development, but this study implies that contribution to effective coordination across teams in large-scale projects may be another important property of test driven development.

In comparison to the guidelines(see section 2.8) for tailoring agile development methodologies to large software systems, proposed by Cao et al. (2004), there are some similarities and some discrepancies between how the project studied in this study operates, and their findings. This suggests that the guidelines provided by Cao et al. (2004) may not apply to all large-scale software development projects. This implies that the guidelines should be re-evaluated, or eventually look at what differences in context factors, which provides a foundation to when the guidelines are applicable.

Both this case and Paasivaara et al. (2008) found that running the daily scrum meetings in a sequential order, had a positive impact on the project. This suggests that this practice could be added to future guidelines for large-scale agile development projects.

6.4.3 Implications for the target audience

Section 1.4 presents four types of target audiences for this report. Computer science students, researchers, practitioners and customers of software development. This subsection will look at what implication this study has for the different types of audiences.

For practitioners and customers of software development, this study shows that agile practices provide valuable coordination in large-scale software development projects. It also shows how certain practices has been modified to better suit the challenges faced when the size of the development project increases.

For researchers, this study implies that more research is needed on this field. It should also give a good starting point for researchers interested in further increasing the knowledge on how agile development can apply to large-scale software development projects.

For computer science students, this report should provide valuable knowledge in how an agile development can be applied in a large scale project, and also shows the importance of the practices and frameworks learnt at university.

6.5 Evaluation of the study

In this section an assessment of the research process as a whole is done by the author, where challenges and rooms for improvement is presented. In addition the trustworthiness of the study is assessed according to a set of pre-defined principles.

6.5.1 The research process

While doing the analysis of the data material gathered, there was a lot of information obtained and understood that could be used for further examination and understanding of the coordination mechanisms within the project. It would therefore have been beneficial to conduct the study in iterations, and preferable over a longer time period, since this would have yielded even richer and deeper data material.

One of the major challenges experienced while conducting the study was to differentiate between process improvement and researching the current state. While conducting the interviews, there were interesting topics that emerged, which were not related to coordination, but still relevant for anyone interested in improving the process. This implies that certain data might be valuable to the company, but not interesting for this particular research. In order to strike a balance, these topics were included in the data material, but not included in the thesis. Since the data material is held confidential, the author has agreed to give a presentation of the report, which will also include the topics not included in this thesis.

6.5.2 How agile is it?

This case study set out to find out how practices used in large-scale agile development affect coordination. Therefore it must also be addressed how agile the project actually is. Officially it is scrum that is the chosen method for development, and the scrum practices are present in the project along with other agile practices such as pair programming and test driven development. At the same time, there are also tools such as creating a critical path diagram, breaking the entire delivery into partial deliveries and creating a critical path diagram, that reduces the agility and implies a certain degree of waterfall methodology. In addition to this, there are factors, such as using kanban, a high focus on continuous improvement and continuous delivery, that implies a more lean methodology. Due to these factors, one can not say that the case studied is a strictly agile project, but rather a combination of agile, lean and waterfall.

6.5.3 Generalisation

It is relevant to see to what extent this study can be generalised for other projects. It is important to note that this is a re-engineering of an old software system, which greatly contributes to creating well-defined requirements, compared to a project which develops a new system from the ground up. In addition it must be taken into account that everyone, working on the supplier side, works for the same consulting company. Many of the big organisations in need of large-scale systems, already have some sort of IT system already implemented, so it is reasonable to say that re-engineering is common when developing large-scale systems, and therefore the findings of this study should be applicable to several large-scale development projects.

6.5.4 Trustworthiness

The evaluation of this study is based upon seven principles proposed by Klein and Myers (1999), for evaluating interpretive field research. The goal for this evaluation is to establish trustworthiness of the study. In the following section, the seven principles are presented with a description based upon Klein and Myers (1999) article, followed by a description of how the principle is applied on this thesis.

The Fundamental Principle of the Hermeneutic Circle

This principle suggests that human understanding is achieved by iterating between considering the independent meaning of parts and the whole that they are from. This is a fundamental principle to interpretive work, and works as meta-principle upon which the following six principles are based upon.

The Principle of Contextualization

There is an inevitable difference in understanding between the interpreter and the author of a text that is created by the historical distance between them. To negate this difference, critical reflection of social and historical background of the research setting is required. This principle has been applied to the thesis through chapter 2 and 4, which are devoted to background information and existing theory on both software development and coordination. A description of the case is also presented in section 4.3.3, where the reasoning for research design is explained by describing the contextual background that the decisions were made upon.

The Principle of Interaction Between the Researchers and the Subjects

Since the research material is socially constructed through interaction between researcher and participants, there must also be critical reflection upon the researchers part in the context. In order to apply this principle to the research, chapter 4.5 describes the previous knowledge and experience of the author, as well as relations that may affect the researchers point of view.

The Principle of Abstraction and Generalization

Unique instances and ideas can be related to ideas and concepts that apply to multiple situations. Validity does not depend on representative cases, but on the plausibility and cogency of the logical reasoning used in describing the results from the case, and in drawing conclusions from them In chapter 4, the logical reasoning used to describe results from the case is presented with the help of pre-existing theoretical frameworks. An evaluation for the generalisation of the case is also presented in section 6.4.1.

The Principle of Dialogical Reasoning

This principle requires the researcher to confront his prejudices which guided the original research design, with the data that emerge through the research process. One benefit and challenge when conducting this research was the limited experience with agile development and coordination in practice. This means that there is little existing prejudice, beyond the theory presented in the earlier chapter. However, in order to further apply the principle, the principle of dialogical reasoning, the thesis has been reviewed by both the supervisor, a fellow student and an external non-technical person, in order to make sure the results discussion and conclusion does not follow a predetermined opinion, but rather reflects the actual results.

The Principle of Multiple Interpretations

This principle requires sensitivity to possible differences in interpretations among the participants as are typically expressed in multiple narratives or stories of the same sequence of events under study. A major benefit with semi-structured interviews, was that they allowed to further investigate issues and statements brought up in previous interviews. Through this, the principle of multiple interpretations were applied to the study. Although there were very few contradicting viewpoints, the different views were presented whenever a contradiction existed in the chapter 5.

The Principle of Suspicion

The principle of suspicion refers to sensitivity to possible biases in the narratives collected from the participants. In this thesis, the statements were not taken at face value. In the analysis, the general consensus of management was compared to the developers to see if there were a conflict of interests, however no such conflicts were discovered.

6.6 Summary

As this study is an exploratory study, it was never intended to create a model for coordination in large-scale agile development projects, but rather increase the knowledge on a complex field. The case studied has been able to maintain a high degree of horizontal coordination, with the exception of one practice that works very vertical. In table 6.1, the different practices discussed in this chapter is presented along with a short description of how they works and what type of dependency the practice handles.

Practice	**How it affects cross-team coordination**	**Dependency type**
Critical path diagram	Greatly reduces the need for cross-team coordination and maps the known dependencies	Known dependencies.
Test driven development	Improves the cross-team coordination and identifies unknown dependencies	Unknown dependencies.
Co-location	Improves coordination at the cost of increased noise level	Both known and unknown dependencies
Scrum of scrums	Distributes tasks to avoid cross team dependencies.	Both types, but primarily known dependencies.
Architect forum	Improves coordination between the architects	Both known and unknown dependencies
Standup meeting	Improves cross-team dependencies, especially when there are known dependencies	Both types, but primarily known dependencies.
Kanban	Creates transparency which allows for easier coordination	Known dependencies
Retrospective meetings	Indirectly improves coordination through process improvement	Both known and unknown dependencies

Table 6.1: How does practices used in large-scale agile development affect coordination

Chapter 7

Conclusion

This master thesis aimed to explore and improve the understanding of how coordination is done through different practices in a large-scale agile project, which is a field with little literature. The conclusion is based upon the discussion presented in the previous chapter. Further work is presented as directions on what could be done in order further improve the understanding of a complex topic.

7.1 Main conclusion

As presented in the discussion in chapter 6, there are several factors and practices, both agile and non-agile that has a positive effect on coordination across teams.
Most of the agile practices on the project had a direct positive impact on the coordination, but there was also one non-agile practice that had a very positive effect.
The most important practices that contributed to a solid coordination tasks across teams were:

- **Critical path diagram**, which helped to distribute the tasks among the teams in order to minimize the amount of cross-team dependencies.
- **Test driven development**, which helped prevent conflicting code across teams, and discover unknown cross-team dependencies.

Other practices that also helped improve the coordination were *Feature teams,*

Co-location, Scrum of scrums, Architect forum, Standup meetings on different times, Kanban and Retrospective meetings.
The need for a non-agile practice and the positive impact it has on coordination also implies that keeping a project completely agile, while scaling it up, can be challenging.

7.2 Further work

Both coordination and large-scale agile projects are in themselves complex topics, so further studies on coordination in large-scale projects are needed. It would be interesting to compare the results of this study to other large-scale agile projects, to see what similarities and differences in practices used in order to achieve good coordination. A comparative study could be very valuable as it could produce "best practices" lists of how to achieve good coordination in large-scale agile projects. In addition, the case studied in this thesis did not follow a pre-defined agile methodology, so it would be interesting to conduct a research to see to what extent large-scale projects actually follow the agile principles and practices.
As test driven development had a very positive impact on coordination in this project, it would be interesting to study the impact of test driven development on coordination in another large-scale development project, in order to see if this could be an added benefit to test driven development.
In this study, the open landscape had a positive impact on coordination, but with a negative side-effect of noise. It would be very interesting to further study the impact of open landscape in large-scale projects in order to see if the noise to coordination ratio is actually worth it.

Bibliography

Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods: Review and analysis, 2002.

D.J. Anderson. *Kanban*. Blue Hole Press, 2010. ISBN 9780984521401.

David Avison and Guy Fitzgerald. *Information systems development: methodologies, techniques and tools*. McGraw Hill, 2003.

Kent Beck. Embracing change with extreme programming. *Computer*, 32 (10):70–77, 1999.

Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

Kent Beck and Cynthia Andres. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.

Barry Boehm. Get ready for agile methods, with care. *Computer*, 35(1): 64–69, 2002.

Fred P. Brooks, Jr. The mythical man-month. *SIGPLAN Not.*, 10(6):193–, April 1975. ISSN 0362-1340. doi: 10.1145/390016.808439.

Lan Cao, Kannan Mohan, Peng Xu, and Balasubramaniam Ramesh. How extreme does extreme programming have to be? adapting xp practices to large-scale projects. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, pages 10–pp. IEEE, 2004.

John Child. Predicting and understanding organization structure. *Administrative Science Quarterly*, 18(2), 1973.

Luisanna Cocco, Katiuscia Mannaro, Giulio Concas, and Michele Marchesi. Simulating kanban and scrum vs. waterfall with system dynamics. In *Agile Processes in Software Engineering and Extreme Programming*, pages 117–131. Springer, 2011.

Alistair Cockburn. Selecting a project's methodology. *IEEE Softw.*, 17(4): 64–71, July 2000. ISSN 0740-7459. doi: 10.1109/52.854070.

Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

Torgeir Dingsøyr and Nils Brede Moe. Research challenges in large-scale agile software development. *ACM SIGSOFT Software Engineering Notes*, 38(5):38–39, 2013.

Torgeir Dingsøyr, Tor E Fægri, and Juha Itkonen. What is large in large-scale? a taxonomy of scaling in agile software development. *Work in progress*, 2013.

Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9): 833–859, 2008.

Jutta Eckstein. *Agile Software Development in the Large: Diving Into the Deep*. Dorset House Publishing Co., Inc., 2004.

H Erdogmus, M Morisio, and M Torchiano. On the effectiveness of the test-first approach to programming. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 31(3):226–237, MAR 2005.

Bent Flyvbjerg and Alexander Budzier. Why your it project might be riskier than you think. *Harvard Business Review*, 89(9):23–25, 2011.

Robert D Galliers and Frank F Land. Viewpoint: choosing appropriate information systems research methodologies. *Communications of the ACM*, 30(11):901–902, 1987.

John Gerring. What is a case study and what is it good for? *American political science review*, 98(2):341–354, 2004.

Jane Frances Gilgun. *Definitions, methodologies, and methods in qualitative family research.* Sage Publications, Inc, 1992.

A Paul Hare. *Handbook of small group research..* Free Press, 1976.

Igor T Hawryszkiewycz. *Introduction to systems analysis and design.* Prentice Hall PTR, 1994.

Ville T Heikkilä, Maria Paasivaara, Casper Lassenius, and Christian Engblom. Continuous release planning in a large-scale scrum development organization at ericsson. In *Agile Processes in Software Engineering and Extreme Programming*, pages 195–209. Springer, 2013.

John K Hemphill. Relations between the size of the group and the behavior of superior leaders. *The Journal of Social Psychology*, 32(1):11–22, 1950.

Rashina Hoda, James Noble, and Stuart Marshall. Organizing self-organizing teams. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 285–294. ACM, 2010.

James E Kelley Jr and Morgan R Walker. Critical-path planning and scheduling. In *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*, pages 160–173. ACM, 1959.

Heinz K. Klein and Michael D. Myers. A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Q.*, 23 (1):67–93, March 1999. ISSN 0276-7783. doi: 10.2307/249410.

Robert E Kraut and Lynn A Streeter. Coordination in software development. *Communications of the ACM*, 38(3):69–81, 1995.

Craig Larman. *Agile and Iterative Development: A Manager's Guide.* Addison-Wesley Professional, 2004.

Craig Larman and Bas Vodde. *Scaling lean & agile development: thinking and organizational tools for large-scale Scrum.* Pearson Education, 2008.

Eric C. Lee. Forming to performing: Transitioning large-scale project into Agile. In *AGILE 2008, PROCEEDINGS*, pages 106–111, 2008. AGILE 2008 Conference, Toronto, CANADA, AUG 04-08, 2008.

Kim Man Lui and Keith CC Chan. Pair programming productivity: Novice–novice vs. expert–expert. *International Journal of Human-computer studies*, 64(9):915–925, 2006.

R. Lyon and M. Evans. Scaling up pushing scrum out of its comfort zone. In *Agile, 2008. AGILE '08. Conference*, pages 395–400, Aug 2008. doi: 10.1109/Agile.2008.19.

Thomas W Malone and Kevin Crowston. What is coordination theory and how can it help design cooperative work systems? In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, pages 357–370. ACM, 1990.

Thomas W Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*, 26(1):87–119, 1994.

Martin N Marshall. Sampling for qualitative research. *Family practice*, 13 (6):522–526, 1996.

N.E.D.D. Miller. *The effect of group size on decision-making discussions.* University of MICHIGAN, 1952.

Henry Mintzberg. Structure in 5's: A synthesis of the research on organization design. *Management science*, 26(3):322–341, 1980.

Deepti Mishra, Alok Mishra, and Sofiya Ostrovska. Impact of physical ambiance on communication, collaboration and coordination in agile software development: an empirical evaluation. *Information and Software Technology*, 54(10):1067–1078, 2012.

Nils Brede Moe, Torgeir Dingsøyr, and Tore Dybå. A teamwork model for understanding an agile team: A case study of a scrum project. *Information and Software Technology*, 52(5):480–491, 2010.

Joe Nandhakumar and Matthew Jones. Too close for comfort? distance and engagement in interpretive information systems research. *Information Systems Journal*, 7(2):109–131, 1997.

Sridhar Nerur and VenuGopal Balijepally. Theoretical reflections on agile development methodologies. *Communications of the ACM*, 50(3):79–83, 2007.

Briony J Oates. *Researching information systems and computing.* Sage, 2005.

Maria Paasivaara, Sandra Durasiewicz, and Casper Lassenius. Distributed agile development: Using scrum in a large project. In *Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on*, pages 87–95. IEEE, 2008.

Frank Padberg and Matthias M Muller. Analyzing the cost and benefit of pair programming. In *Software Metrics Symposium, 2003. Proceedings. Ninth International*, pages 166–177. IEEE, 2003.

Matjaž Pančur and Mojca Ciglarič. Impact of test-driven development on productivity, code and tests: A controlled experiment. *Information and Software Technology*, 53(6):557–573, 2011.

Minna Pikkarainen, Jukka Haikara, Outi Salo, Pekka Abrahamsson, and Jari Still. The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13(3):303–337, 2008.

M. Poppendieck and T. Poppendieck. *Lean Software Development: An Agile Toolkit.* The Agile software development series. Addison-Wesley, 2003. ISBN 9780321150783.

Mary Poppendieck. Lean software development. In *Companion to the Proceedings of the 29th International Conference on Software Engineering*, ICSE COMPANION '07, pages 165–166, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2892-9. doi: 10.1109/ICSECOMPANION.2007.46.

Mary Poppendieck. Principles of lean thinking. *IT Management Select*, 18, 2011.

Lene Pries-Heje and Jan Pries-Heje. Why scrum works: A case study from an agile distributed project in denmark and india. In *Agile Conference (AGILE), 2011*, pages 20–28. IEEE, 2011.

W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press. ISBN 0-89791-216-0. URL `http://dl.acm.org/citation.cfm?id=41765.41801`.

D. Rubinstein. Standish group report: Theres less development chaos today. *Software Development Times*, 1, 2007.

K. Schwaber and M. Beedle. *Agile software development with scrum.* Series in agile software development. Prentice Hall, 2002. ISBN 9780130676344.

Ken Schwaber. Scrum development process. In *Business Object Design and Implementation*, pages 117–134. Springer, 1997.

Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *Software Engineering, IEEE Transactions on*, 25(4):557–572, 1999.

Helen Sharp and Hugh Robinson. An ethnographic study of xp practice. *Empirical Software Engineering*, 9(4):353–375, 2004.

Diane E Strode, Sid L Huff, Beverley Hope, and Sebastian Link. Coordination in co-located agile software development projects. *Journal of Systems and Software*, 85(6):1222–1238, 2012.

J.D. Thompson, W.R. Scott, and M.N. Zald. *Organizations in Action: Social Science Bases of Administrative Theory.* Transaction Publishers, 1967.

Bruce W Tuckman. Developmental sequence in small groups. *Psychological bulletin*, 63(6):384, 1965.

Andrew H Van de Ven. *Group decision making and effectiveness.* The Comparative Administration Research Institute of the Center for Business and Economic Research, Kent State University Press., 2007.

Andrew H Van de Ven, Andre L Delbecq, and Richard Koenig Jr. *Determinants of coordination modes within organizations.* JSTOR, 1976.

Guus Van Waardenburg and Hans Van Vliet. When agile meets the enterprise. *Information and Software Technology*, 55(12):2154–2171, 2013.

Geoff Walsham. Doing interpretive research. *European journal of information systems*, 15(3):320–330, 2006.

Robert K Wysocki and Rudd McGary. *Effective project management: traditional, adaptive, extreme.* John Wiley & Sons, 2003.

Robert K Yin. *Case study research: Design and methods*, volume 2. sage, 1984.

Appendix A

Interview guide

A.1 Interview guide for management

Background information

- What is your name?
- What is your role in this project?
- How many years of IT experience do you have?
- What is your educational background?
- What are your previous experience with agile development?

Agile development

- What practices, agile or not, do you think have been particularly useful in this project, and what makes them useful?
- Which challenges has come up due to the choice of development method?
- An important aspect of agile development is that work should always be done on the most important tasks. Do you feel that this is still the case with so many people working on the same project?
- Has the size of the project affected how you work, compared to previous experiences?

Coordination

- How is coordination done in this project?
- What do you think makes/doesn't make this project a well-coordinated project?
- How should members of the development teams proceed if they need information from a different team?
- Are there members of the project that has key knowledge or competence, whose role can not be filled by others on short notice?
- Are there any bottlenecks within the project?

Boundary spanning

- How are tasks handled when interaction with people outside the project is required?
- What happens when information is needed from the customer?

Other

- Would you recommend a similar development method to other projects?
- Anything else to add?

A.2 Interview guide for developers

Background information

- What is your name?
- What is your role in this project?
- How many years of IT experience do you have?
- What is your educational background?
- What are your previous experience with agile development?

Dependencies

- What are some typical dependencies in this project?

- How are there reported?

Agile development

- What practices, agile or not, do you think have been particularly useful in this project, and what makes them useful?
- How is coordination done in this project?
- What do you think makes/doesn't make this project a well-coordinated project?
- Which challenges has come up due to the choice of development method?
- An important aspect of agile development is that work should always be done on the most important tasks. Do you feel that this is still the case with so many people working on the same project?
- Has the size of the project affected how you work, compared to previous experiences?

Information flow

- How would you proceed if yuo need information from someone in a different team?
- How much do you know of what is happening within the other teams?
- How do you ensure that the work you're doing doesn't affect development in other teams
- How does working in an open office structure affect how you work?
- Are there members of the project that has key knowledge or competence, whose role can not be filled by others on short notice?
- Are there any bottlenecks within the project?

Boundary spanning

- How are tasks handled when interaction with people outside the project is required?
- What happens when you need information from the customer?

- Do you feel there are tasks that are delayed because you were waiting on someone outside the project?

Other

- Would you recommend a similar development method to other projects?
- Anything else to add?

Index