Process Tailoring in Large-Scale Agile Programs

A case study of coordination in Autonomous DevOps Teams

Andreas Aasheim

Thesis submitted for the degree of
Master in Software Engineering
60 credits

Department of Informatics
Faculty of Mathematics and Natural Sciences
UNIVERSITY OF OSLO

Spring 2018

Process Tailoring in Large-Scale Agile Programs

A case study of coordination in Autonomous DevOps Teams

Andreas Aasheim

Spring 2018

2018

Process Tailoring in Large-Scale Agile Programs

Andreas Aasheim

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Background: Teams in large-scale agile programs need to achieve collaborative software development. A proposed guide to collaboration is effective coordination. Large-scale agile software development is well accepted in the software industry, but there is little understanding of such projects and programs achieve effective coordination in autonomous cross-functional teams. Therefore, I conducted a case study of a large-scale software program consisting of seven autonomous teams in an organization. Five of them was DevOps teams, where DevOps is merely a team composed of developers who are working on the development and operational tasks.

Aim: The thesis aims to investigate what dependencies and their related agile practices that act as coordination mechanisms to facilitate the large-scale agile development. Additionally, the aim is also to recommend a starter set for providing coordination in the large-scale by using a dependency taxonomy, which mapping agile practices.

Method: A qualitative case study was conducted for the research design. The data was collected by conduction 40 observed meetings, as well as 18 entire working days in the organization's open work area to observe them in their everyday work. A dependency taxonomy was used to map and categorize the coordination mechanisms.

Results: The results revealed that there were 34 coordination mechanisms and 77 pairs of dependencies presented in the program. The coordination mechanisms could be mapped into three categories, with subcategories each: *knowledge* dependency, *process* dependency, and *resource* dependency. The *knowledge* dependency was predominant with the frequency of 73 % of the three categories. These means that focusing on selecting agile practices that address the types of *knowledge* dependency should be recommended for providing coordination. Furthermore, the results revealed that 12 agile practices would be a good choice for coordinating and tailoring a large-scale program.

Conclusion: It is possible to use a dependency taxonomy to map coordination mechanisms in a large-scale agile program. The coordination mechanisms made collaboration between the teams in the program, by implement Scrum of Scrum meetings, daily stand-up meetings, demo meetings, Sprint Planning meetings, and introduce different roles, such as project managers, team leads, Product Owners and DevOps developers. These mechanisms lead to fast Sprint periods, frequent production setting, a common understanding of what is being created, and autonomous decisions in the program.

Acknowledgements

First, I would like to thank all those who have contributed to my master thesis, both in terms of academic and patient support. I would like to thank my supervisor Viktoria Stray from the University of Oslo for all your help, guidance and putting me in contact with the relevant people. I am also thankful to Nils Brede Moe from the research group in SINTEF for giving me the opportunity to write this thesis, for perfect feedback and guidance throughout this thesis. Furthermore, I am thankful to all the participants in the study, for welcoming me into their organization and therefore I would like to give a special thanks to the CTO in Knowit Objectnet AS, Jan Henrik Gundelsby for including me in their projects and workplace.

A special thanks go to my fellow students and other employees from the research group Programming and Software Engineering at the Department of Informatics for discussions and other valuable help.

Lastly, I would like to thank my family and friends for their help and support, my parents Bente and Tom, for all great contributions and my siblings for continues support. I am especially thankful to my dear Frida for her love, encouragement, and optimism through this research and writing period.

Oslo, May 2018

Andreas Aasheim

Table of Contents

List of Figures

List of Tables

1 Introduction

Firms today must be able to adapt to complex and unpredictable tasks in IT projects, where it is necessary to quickly change the focus on tasks according to the costumer's needs. Autonomous cross-functional teams, teams put together with different expertise, are increasingly being used in IT projects, and are often relevant in large-scale projects and distributed projects (Marczak & Damian, 2011). Autonomous cross-functional teams exploit skills across functions, are assumed to make better collaboration, and the decision-making in such teams are more divided. Cross-functional teams are spreading fast in organizations as they attempt to improve coordination, and are most often structured as work-groups or teams, created to make decisions lower in an organization's hierarchy (Denison, Hart, & Kahn, 1996). This creates a smoother structure with increased involvement of responsibility across the different teams and more effective decisions.

This master thesis aims at making a comparative study of coordination mechanisms of the future's flexible agile software development. This thesis is linked to a research project led by SINTEF. To clarify and define the scope of this thesis, when the terms cross-functional teams are being mentioned, it is referred to each their respective units within a product producing company. This master thesis is therefore about in-house development.

Furthermore, since I have chosen to look at autonomous *DevOps* teams, I will study the notion of coordination of teams into how a combination between development and operations (DevOps) people in large-scale development can be integrated as an interdisciplinary team. Coordination is achieved through coordination mechanisms and dependencies for choosing effective coordinative practices to better support a collaborative systems development environment (Strode, 2016), such as different types of meetings and roles. The dependencies in agile software development can help teams, team members, and other participants involved in development projects to choose valid coordinative practices (Strode, 2016). In this thesis, I will examine coordination through studying the different dependencies and their associated agile practices acting as coordination mechanisms in a large-scale agile program which practices the *DevOps* culture.

1.1 Motivation

In my daily work and for me personally, collaboration and coordination are important. From my experience in the airline industry, I learned that collaboration in teams is a key factor and provides success to get the airplanes on time for the commercial airlines at Norway's largest international airport. Cooperation and coordination provides motivation and makes a good form for research on collaboration and coordination in teams, especially in the software industry. This master thesis is, therefore, a study about what type of mechanisms that form good collaborated teams.

Moreover, cross-functional teams are often characterized by complex processes and high uncertainty (Parker, 2003). To make cross-functional teams in companies effective in today's and future value creation processes, there is a need for knowledge of how such teams work. It is necessary to find out how autonomous cross-functional teams can work in software development projects and test new models for team organization, where the skills and benefits of a working culture are utilized in the best possible way tailored by coordination mechanisms.

Today software is present in consumer and business products, in cars, airplanes, smartphones, and people regularly use software-based products home or at work. More than two billion people using the broadband internet today. Of this, new markets emerge for software companies challenging established market incumbents with software-based products (Schmidt, 2016). Agile development is an answer to the establishment of new markets in the software industry, and new technologies have quickly adopted since the beginning of the 2000s. The technological potential led to heavy investment into the IT industry, and more and more software applications were now developed for the customer market (Schmidt, 2016). At the beginning of the agile software development period, it was started by cross-functional teams (Parker, 2003). Cross-functional teams are sweeping across organizations today and this master thesis is studying how participants as developers, testers, integration designers, project managers, and designers are gathered and put together into a team for better collaboration.

Nowadays, there is a need for a general approach to software development when it comes to collaboration between teams (Marczak & Damian, 2011; Parker, 2003; Wiedemann, 2018). A description is the key concept of "flow" from the Lean development method (Fitzgerald & Stol, 2017), the goal was to achieve "flow" between the various actors. These

actors are usually in teams in an organization and is an effect on how the organization is coordinated. This organizations are deciding to move from traditional plan-driven software development to agile approaches. The reason for that is so the organization can stay competitive. Therefore, the agile approaches have been deciding to implement cross-functional *DevOps* teams (Wiedemann, 2018). This collaboration is a combination between development and operations (DevOps) people. *DevOps* has some perspectives; a culture of collaboration between team members, automation of build, deployment and testing, and sharing of knowledge between the teams (Bang, Chung, Choh, & Dupuis, 2013).

Such collaboration has been proposed in research already, including governance, architecture, the support of knowledge, skills, and abilities, issues and industrial challenges in cross-functional *DevOps* teams. Meanwhile, there is still few studies of collaboration in large-scale projects and its coordination of cross-functional *DevOps* teams. Findings in the *DevOps* environment has been expressed in research in various ways (Bang et al., 2013; Nitto, Jamshidi, Guerriero, Spais, & Tamburri, 2016; Wiedemann, 2018).

There seems to be a gap in research when it comes to studies based on coordination between development and operations and in large-scale agile programs to enable a good collaboration culture. According to Wiedemann (2018, p. 4931), "the IT organizations recognize that they have to shift from traditional service-provider role to agiler oriented approaches to become a partner for the business." The study by Lwakatare et al. (2016), found that the application of DevOps concepts to the embedded systems domain underscored the importance of agile software development and specifically cross-functional teams.

Another statement is that coordination needs change over time and there are agile practices that are emerged and disappeared, and a change from scheduled meetings to unscheduled meetings occurs. "Coordinating mechanisms are dynamic social practices that are under continuous construction" (Jarzabkowski, Lê, & Feldman, 2012). Strode (2012, p. 15) outlines that "one of the goals of a system development methodology is to provide ways to organize and coordinate development." Lastly, Dingsøyr, Moe, and Seim (2018) suggest that future work should seek to develop a further understanding regarding coordination mechanisms in large development programs to investigate how coordination mechanisms are tailored to the specific context of a program. I would argue that these statements and gaps increase the need for this master thesis.

1.2 Research Area and Questions

The research area of this thesis is autonomous cross-functional teams created by coordination mechanisms to address dependencies between development and operations (DevOps), stakeholder, and supplier within a large-scale program in agile software development. The process of this aspects will be studied by examining agile practices that act as coordination mechanisms, by daily stand-up meetings, Scrum of Scrum meetings, project meetings, and other relevant mechanisms that are dependent on the agile software development methodology.

In this thesis the overall research focus will be to study the effects of dependencies and coordination mechanisms in autonomous *DevOps* teams for an in-house software company. Following research questions are:

RQ1: What dependencies and their associated agile practices that act as coordination mechanisms facilitate the large-scale agile development?
RQ2: What could be a recommended starter set for providing coordination in large-scale agile development programs by using a dependency taxonomy?

1.3 Approach

I will conduct a case study to answer the research questions with theories from other case studies. Datasets through observations in an in-house software company will be used to do the case study, and these methods will be evaluated at the team level. The software company under study consists of seven teams, all of them will be studied.

The workflow that creates results and to evaluate data I will choose a strategy for the data analysis. To conduct the datasets and data analysis, a software application called NVivo[1] was used. The software application will convert the dataset into more deliberate data. This topic is presented more in Chapter 3 Research Method and Design.

[1] NVivo is a registered trademark of QSR International, www.qsrinternational.com

1.4 Chapter Overview

Chapter 2: Background and Theoretical Perspective gives a brief introduction to *agile software development methodologies, large-scale agile development, coordination, autonomous cross-functional teams* and the *DevOps* concept that is considered to be necessary to understand the rest of this thesis.

Chapter 3: Research Method and Design introduces and explains the research methods used in this work to study coordination mechanisms and agile practices in autonomous cross-functional teams consisting of development and operations in the large-scale agile program.

Chapter 4: Research Context presents an overview of the large-scale agile program in the organization under study to provide a context for the study.

Chapter 5: Results presents the results related to the methods and findings in coordination mechanisms as agile practices mapped to dependencies, as well an extensive description of some one of them.

Chapter 6: Discussion presents a discussion on the results from the case study against the findings from prior research and the research questions from this work.

Chapter 7: Conclusion and Future Work contains the conclusion to the research questions of this thesis and points at interesting directions for future work.

2 Background and Theoretical Perspective

This chapter introduces important background theory to this research. First, there will be a brief introduction to *agile software development* methodologies, with methods such as *Scrum*, *Lean* and *Kanban*. Then, I introduce methodologies to *large-scale agile development*. Then, theories on *coordination* are presented with a brief introduction to *autonomous cross-functional teams*. Lastly, there will be a brief introduction to the *DevOps* concept. To be able to discuss the results of this study, it is important to look at relevant cases and theory which can put this into perspective.

2.1 Agile Software Development

This thesis uses theories on agile software development, such as theories and studies on the Scrum framework, lean software development, and Kanban. Software development is constantly evolving due to changes technologies and new demands from users (Nerur, Mahapatra, & Mangalaraj, 2005). Software development is a knowledge-intensive activity and belongs to the engineering discipline with an engineering effort involving a lot of design, and the production is relatively simple (Wohlin, Šmite, & Moe, 2015). This description is called software engineering. The dynamic business environment has given rise to emergent organizations that continuously adapt their structures strategies and policies to suit the new technical environment (Nerur et al., 2005). A theory on this suggests that software engineering is a balancing act between three resources; human, social and organizational (Wohlin et al., 2015).

Furthermore, software engineering was coined in 1968 at a conference whose aim was to discuss the need for the software development discipline (Kirk & MacDonell, 2015). This, to be more strongly based on theoretical engineering principles. Software development is the production of software, which consists of a sequence of fundamental activities called a "software process" (Paulk, Curtis, Chrissis, & Weber, 1993). The first software process model was the traditional waterfall model. This model, a then-popular model used in manufacturing, was adopted as the standard approach for developing software. As time progressed, it became apparent that a strict implementation of this model was not

appropriate for software. A number of modifications, for example, extreme programming, have emerged (Kirk & MacDonell, 2015).

The main goal of the traditional is to plan in early stages to ensure design flaws before programming is started, thus we get a plan-driven method (Munassar & Govardhan, 2010). Moreover, as a response to the plan-driven process model, the agile development methods emerged. From the field of agility in IT, Fink and Neumann (2007, p. 444) define agility as "the ability to respond operationally and strategically to changes in the external environment. The response has to be quick and effective for the organization to be considered agile" (Fink & Neumann, 2007). The agile methods were created because there was a need for methods to take into account the unpredictability of the world, including the higher rates of change and to involve the customer much earlier during development (Dybå & Dingsøyr, 2008). One other definition is given by Erickson, Lyytinen and Siau (2005, p. 89) and define agility as fellows: "agility means to strip away as much of the heaviness, commonly associated with the traditional software development methodologies, as possible to promote quick response to changing environments, changes in user requirements, accelerated project deadlines and the like."

The agile methodology has become a key factor in driving innovation and gaining a competitive advantage in the digital age. Coyle, Conboy, and Action (2015) consider that one of the most differences between organizations that follow agile approaches and organizations that follow more traditional approaches is that the agile ones establish autonomous, self-organized teams in their projects (Coyle et al., 2015). These self-organized teams are an answer to that the IT companies must be able to adapt complex and unpredictable tasks in IT projects, where it is necessary to quickly change the focus on tasks according to the costumers needs.

2.1.1 Scrum

Scrum is a framework used in the agile software development process to organize teams and get work done. Scrum allows teams to choose the size of work to be done and decide themselves how best to do it by a "lean" approach to software development (Sutherland & Schwaber, 2007). In Scrum, a working period called Sprint is conducted. A Sprint is a period which the Scrum team works one month or two weeks. During one Sprint the Scrum team produces a product increment (Schwaber & Beedle, 2001).

A recently study by Cooper and Sommer (2018) found at the beginning of each Sprint in six different case studies from six companies in North America and Europe, the development teams met to agree on what it can accomplish in the sprint and created a task plan by a Sprint Planning meeting. During the Sprint, daily stand-up was held to ensure that work is on course to accomplish in the last 24 hours, and what should be done in the next 24. At the end of each Sprint, product demo meetings and retrospective meetings were held to review how team members worked together.

Moreover, to implement the Sprint, tasks for the working period is put into a Product Backlog. The Product backlog "contains a list of prioritized tasks defined by the Product Owner. The development team breaks this backlog into sprint backlog items and tracks its progress during each Sprint" (Schmidt, 2016, p. 17). Furthermore, the Scrum framework based on Sutherland and Schwaber (2007) has a list of different practices shown in Table 1:

Table 1: Practices in the Scrum framework

Type	**Practice**
Roles	Scrum team Scrum Master Product Owner Test lead
Ceremonies	Sprint Planning meeting Sprint Review meeting Scrum of Scrums meeting Daily Scrum meeting Product demo meeting

Scrum Team

The Scrum team organizes itself and they consist of seven people, plus/ minus two members. A team is a unit of people which produces the software and selects the Sprint goal and specifies work results (Sutherland & Schwaber, 2007) The team is cross-functional and should include the necessary roles to complete their tasks (Schwaber & Beedle, 2001). Roles in Scrum teams can be defined as project manager, team lead, test lead, developers, Scrum Master and Product Owner (Beranek, Zuser, & Grechenig, 2005; Evbota, Knauss, & Sandberg, 2016).

Scrum Master (SM)

The SM has the role to ensure that the team is fully functional and productive and enables close cooperation across all roles (Sutherland & Schwaber, 2007). The SM acts as a facilitator for software development teams, to make sure agile practices are followed (Bass, 2014). A study by Bass (2014) in large-scale projects, found that the SM work together in geographically distributed teams and use Sprint Planning to avoid development tasks that overlap team boundaries, coordinate status, and effort across teams. The study identified that the SM role comprises six activities: process anchor, stand-up facilitator, impediment remover, Sprint planner, a Scrum of Scrums facilitator, and integration anchor (Bass, 2014). Another study in large-scale program done by T. Dingsøyr et al. (2018) found that the Scrum Master facilitated daily meetings, iteration planning, demonstration, and retrospective.

Product Owner (PO)

The PO defines the features of the product, decides on release date and content. The PO also accepts or reject works results and can change features and priority every sprint periods. Furthermore, the PO is responsible for the profitability of the product and prioritizes features according to market value (Sutherland & Schwaber, 2007). Findings in studies by Bass (2015) was that the PO's role was to reconcile competing business interests, the PO identifies and prioritizes customer requirement and that the PO was formed into teams. A second finding was that it was identified nine team functions for a PO: groom, prioritize tasks, release master, technical architect, governor, communicator, traveler, intermediary and risk assessor. A second study in this role is done by Bass, Beecham, Nic Canna, Noll, and Razzak (2018). They found that the PO is responsible for gathering and prioritizing requirements and assessing whether features have met the definition of "done". The PO is also responsible for translating business needs into a software implementation.

Test Lead

In the Perform programme by Dingsøyr, Moe, Fægri, and Seim (2018), about exploring software development at the very large-scale, they found that the test lead made sure that testing was conducted at team level by unit tests, integration tests, system tests, and system integration tests.

Sprint Planning Meeting

The work to be performed in the Sprint is planned at the Sprint Planning meetings (Schwaber & Sutherland, 2013). The meeting is organized by the Scrum Master and is a two-phase meeting. First, users, management, the customer, and the Scrum team held the meeting to decide goals for the next sprint. Second, Scrum Master and the Scrum team focusing on how the product increment is implemented during the Sprint (Abrahamsson, Salo, Ronkainen, & Warsta, 2017). In a case study by Paasivaara, Durasiewicz, and Lassenius (2009), they found in distributed projects that the Sprint Planning meetings were divided into three phases; distributed meeting, local meeting onsite and local meeting offsite. The PO presented the prioritized items in the backlog, and the team asked questions. The distributed meetings were arranged using teleconferencing and tools for application sharing. The meetings were time-boxed because of the time-zone difference. The meetings were held at the end of the day for the offsite team and the onsite team continued by dividing the backlog items into more detailed tasks for the rest of the day. The offsite team continued the work the following morning.

Sprint Review Meeting

The Sprint review, or demo, is an informational meeting at the end of a Sprint and the Scrum Master is responsible for conducting it. During the Sprint review, the Scrum team and stakeholders collaborate about what was done in the Sprint (Schwaber & Sutherland, 2013).

Scrum of Scrums Meeting

Scrum of Scrums allows teams to communicate with each other to ensure that the software of each team integrates well with the fundamentals of the other teams. The meeting should last a maximum of 15 minutes (Larman & Vodde, 2010). A previous study in Scrum of Scrums have been identified by Lee & Young (2009) with three distributed models:

- *Isolated Scrums:* Used mainly where outsourcing is used. These teams are not cross-functional and not flexible. There are often teams that are isolated across different geographies. Projects are often offended by communication problems and weak team relationships.

- *Distributed Scrums:* The Scrum team is isolated in several places and integrated by Scrum of Scrums that meets regularly across locations. This model works across cross-functional teams and isolated Scrum teams. The Scrum team is linked with Scrum of Scrums where Scrum Master meets regularly across different geographies.
- *Fully distributed:* The Scrum teams are cross-functional with members from several different locations. This model is suggested for experienced flexible teams in several places because the cost per historical point is the same as it would be for collocated teams.

In the literature, the Scrum of Scrum meetings have been described as the mechanism for managing inter-team coordination in large-scale Scrum, but how to implement it in projects with a higher number of different teams is not explained (Paasivaara, Lassenius, & Heikkilä, 2012). Moreover, Paasivaara et al. (2012) found how Scrum of Scrums meetings was used in two large-scale, globally distributed Scrum projects. Both projects worked with at least 20 Scrum teams, and 58 interviews were conducted by project staff including managers, architects, Product Owners, developers, and testers. Moreover, the results showed that Scrum of Scrums meetings with representatives from all teams was seriously challenged. The audience was too big to keep everyone interested, the participants did not know what to report to the other teams and challenges with coordination at the project level remained (Paasivaara et al., 2012).

Daily Scrum Meeting

The daily Scrum meeting is a 15-minute meeting designed to clarify the state of the Scrum (Sutherland & Schwaber, 2007). The daily Scrum meeting has multiple names, and the most used originates from Extreme Programming (XP) and is *daily stand-up meeting* (Stray, Sjøberg, & Dybå, 2016). A recent study shows that 87 % of those who practice agile methods in their projects use daily stand-up meeting (Stray, Moe, & Bergersen, 2017). Each team member speaks to three Scrum questions (Stray, Moe, & Aurum, 2012):

Q1: What have I done since the last meeting?
Q2: What will be done before the next meeting?
Q3: What obstacles are in the way?

Moreover, Stray et al. (2016) analyzed the data from four countries, 12 software teams, 60 persons and 79 observed daily stand-up meetings and concluded that these types of meetings affect more than we think. The study shows that the meeting should be held on time before lunch. Then the teams can go for lunch together and talk about the topics that were raised in the meeting before lunch. The study also showed that it to very important to be standing during these meetings because their result shows that "the teams that had all participants standing had considerably shorter meetings than those that had some people, especially the Scrum Master, sitting" (Stray et al., 2016).

Product Demo Meeting

Demonstrate the functionalities to the customer is done by a product demo meeting (Jain & Suman, 2017). Nyrud and Stray (2017) found that a demo meeting facilitated coordination because it was an arena for creating common expectations. This meeting also created a common understanding of the finished product. In the case by Paasivaara et al. (2009), they found that the biggest problem with the demo was the used technology, teleconferencing and application sharing, did not offer enough possibilities to communicate efficiently.

2.1.2 Lean Software Development

Lean software development is "all about getting the right things to the right place at the right time the first time while minimizing waste and being open to change" (Raman, 1998, p. C13). This approach was derived from the Lean manufacturing, especially the Toyota production system from 1948, because the Lean methodology was successful in the car manufacturing industry (Poppendieck & Poppendieck, 2003). The main goal in Lean is to maximize the value produced by an organization and delivered to the customer. This is done by finding and eliminating waste, controlling variability and maximizing the flow of delivered software all within the culture of continuous improvements (Anderson, Concas, Lunesu, & Marchesi, 2011). Moreover, when it comes to Lean software development the book from the Poppendiecks outlines that an important concept is to manage workflow with the concept of pull systems, which means that tasks are put in production only when a customer asks for it (Poppendieck & Poppendieck, 2003). "The pull system in software development is short iterations based on customer input at the beginning of each iteration" (Poppendieck & Poppendieck, 2003, p. 74).

Furthermore, Anderson et al. (2011) point out that the pull-based method Kanban has in recent years been introduced more and more to software development. It is becoming one of the keys to Lean practice in software development. The Lean methodology consists of seven fundamental principles (Ahmad, Markkula, & Oivo, 2013): Eliminate waste, build quality in, create knowledge, defer commitment, deliver fast, respect people and optimize the whole.

2.1.3 Kanban

Kanban was introduced in the Toyota production system as a scheduling system for Lean and Just-In-Time production during the late 1940's and in the early 1950's. This scheduling system was conducted to catch up with the American car industry. This method combined with the Lean methodology was a success for Toyota (Ohno, 1988). Recently it has however been more popular in the software development industry and the methodology has seen an increasing amount of project that applies Kanban and Lean principles (Anderson et al., 2011). The Kanban methodology consists of five fundamental principles (Ahmad et al., 2013): Visualise the workflow, limit work in progress, measure and manage flow, make process policies explicit and improve collaboratively.

Moreover, Anderson et al. (2011) investigated the different impact of the Lean-Kanban approach and defined the Kanban as: "The work in process (WIP) is usually made evident to the team, and to the stakeholders, using a Kanban board. In general, we can define the Kanban software process as a WIP limited pull system visualized by the Kanban board" (Anderson et al., 2011, p. 14).

2.2 Large-Scale Agile Development

Large-scale agile development has been used to describe agile development in a range of context (Dingsøyr et al., 2018). In a case study presented by Moe, Olsson and Dingsøyr (2016) large-scale agile development is described on aspects as the number of people involved in the development and the number of teams. Arguments for a definition based on the number of teams in the large-scale agile development is presented by Dingsøyr, Fægri, and Itkonen (2014). In the case study, they present the number between 2-9 teams. In very large-scale Dingsøyr et al. (2018) describe the number of more than ten teams.

However, to be able to discuss the results of this study on large-scale agile development, which will be done in Chapter 5 and 6, it is important to look at a relevant theory which can put this into perspective. The next sections will present more relevant theories and taxonomies that are useful when investigating the findings. The next sections is about guidelines for tailoring agile by Rolland, Mikkelsen, and Næss (2016), coordination in agile software development from different cases, coordination in large-scale by Nyrud and Stray (2017), determinants of coordination dependencies and their mechanisms within the taxonomy by Strode (2016), important advantages on autonomous cross-functional teams by Parker (2003), and the concept on the DevOps methodology by Lwakatare et al. (2016).

2.2.1 Guidelines for Tailoring Agile in Large-Scale

Based on a study from a large-scale agile software development effort involving more than 120 participants in a Governmental organization and running for 3,5 years, Rolland et al. (2016) described a guideline for tailoring agile as illustrated in Table 2:

Table 2: A guideline overview for tailoring agile in large-scale (Rolland et al., 2016)

Guideline	**Description**
Record and move on	Is important to building trust in the development organization, enabling pragmatic decisions and temporary solutions, by not waiting for sorting out contractual details.
Improve inter-team coordination	Establish long-term "communities of practice" and short-term "task forces" term to improve inter-team coordination.
Scale the project	The study implements first the importance to give the customer time to get accustomed to the working process. Second, the importance to give them training activities to ensure customers are aware of what is required of them. Both these points should be done before a ramp-up phase.
Adjust content in sprints	The customer must be given time to absorb and process new information, and coordinate requirement elicitation with stakeholders in their organization. This practice can being followed by having technical Sprints, where the customer is left alone, and the developers focus on technical tasks.
Experiment with new practices	Projects should experiment with practices that highlight functional and technical interdependencies in the software being developed for tailoring agile. This will improve coordination and communication across teams and roles.
Demo	In the study, demos were improvised in the middle of sprints to get the users feedback on functionality and interaction design. The demos made the communication and collaboration with the customer smooth.

2.3 Coordination

This master thesis is a research on coordination in autonomous DevOps teams in large-scale agile programs. The definition of coordination in organization studies and software development at the organization, project and team level can be seen in different ways (Strode, Huff, Hope, & Link, 2012). Therefore, theories on coordination are outlined here.

2.3.1 Research on Coordination

One proposed theory of coordination was done by Malone (1988), he proposed the definition of coordination as: "when multiple actors pursue goals together, they have to do things to organize themselves that a single actor pursuing the same goals would not have to do" (Malone, 1988, p. 5). He called these extra organizing activities coordination. Moreover, this definition was refined in 1994 by Malone and Crowston and introduced the following definition: "coordination is the managing of dependencies between activities" (Malone & Crowston, 1994, p. 90). This theory is based on ideas from organization theory, management, computer science and economics. However, the idea in this coordination theory is that coordination is needed to address and identifying dependencies, categorizing those dependencies, and identifying the coordination mechanisms in a situation, as stated in Strode et al. (2012).

Coordination in Agile Software Development

In agile software projects coordination has been identified in the empirical research. Pikkarainen, Haikara, Salo, Abrahamsson, and Still (2008) used Malone and Crowston's theory from 1994 to study two small co-located agile projects. In this study, they found Sprint Planning meetings, daily meetings, and open work area. The findings were found to promote communication as a mechanism. Moreover, the product backlog, scrum board, sprint backlog, and daily meetings were findings identified for achieving coordination in a globally distributed Scrum project done by Pries-Heje and Pries-Heje (2011). The study identified coordination as one of the critical elements that explain why Scrum works as a project management project. Other findings are done by Moe, Dingsøyr, and Dybå (2010). In their case study in a co-located Scrum project about how to understanding a teamwork model for an agile team, the findings were that team members "not knowing what others were doing" (Moe et al., 2010, p. 488). In this project, the coordination suffered due to

misapplication of Scrum practices partially caused by an existing organizational structure that promoted specialization of skills within individuals.

Moreover, Strode (2016) explored dependencies in three co-located agile software development projects and organized them into a taxonomy of dependencies. The findings on coordination mechanisms was mechanisms such as cross-team walk, task, product backlog, burndown chart, formal meeting, and Wiki for sorting project information (Strode, 2016). She mapped the findings on coordination mechanisms into categories and drew it in a table based on dependency keys as knowledge, process and resource. Furthermore, coordination mechanisms based on strategy components as *synchronization activity*, *synchronization artefact*, *proximity*, *availability*, *substitutability*, *boundary spanning activity*, *boundary spanning artefact* and *coordinator role*. "This taxonomy provides basic knowledge about dependencies useful for deciding how to assemble practices from commonly used in agile methods to achieve effective project coordination" (Strode, 2016, p. 43).

Coordination in Large-Scale Agile

In large-scale software, development coordination is an important but challenging success factor (Nyrud & Stray, 2017), and coordination has been identified likely as in co-located programs. In large-scale agile, a case study on inter-team coordination found eleven different coordination mechanisms by Nyrud and Stray (2017). The findings were mechanisms as an agile process, open work area, stand-up meeting, retrospective, backlog grooming, demo, Sprint Planning, and Jira. They mapped the coordination mechanisms into five categories and drew it on a framework of coordination mechanisms proposed by Van De Ven, Delbecq, and Koenig (1976). The categories in the framework were used with the type of programming with impersonal mode and the type of feedback with personal mode and group mode.

The framework of Van De Ven et al. (1976) in the case was valuable as a tool to map coordination mechanisms in a large-scale program. The framework was valuable "because the framework makes you aware of what to look for and to understand the concept of coordination" (Nyrud & Stray, 2017, p. 4). Furthermore, the framework was valuable in terms of mapping impersonal modes of coordination such as rules, plans and communication systems (programming).

2.3.2 Theories on Coordination Mechanisms

To examine coordination mechanisms to suggest agile practices, such as co-located customers and short iterations, Cao and Ramesh (2007) used the proposed framework by Van De Ven et al. (1976). Moreover, Malone and Crowston's theory from 1994 was used in the case from Pikkarainen et al. (2008) to study co-located agile projects with findings in communication as a coordination mechanism. The case study from (Strode, 2016) addressed dependencies and coordination mechanisms by a dependency taxonomy in co-located projects. However, since Strode (2016) suggests that further research could assess the applicability of her dependency taxonomy in contexts such as large-scale or distributed agile software development, the approach by Strode (2016) is suitable for my case, and therefore her dependency taxonomy was followed in my case study about a large-scale program.

However, since coordination is achieved through coordination mechanisms, such as meetings and tools for sorting project information, this case will examine coordination through outlining the different mechanisms in large-scale. Moreover, since Strode (2016) also propose a taxonomy for mapping out coordination mechanisms, my case use her proposed dependency taxonomy in the collected data.

Strode (2016) defines coordination in a dependency taxonomy. The taxonomy identifies three modes of dependencies: *knowledge*, *process* and *resource* with subcategories each, as shown in Figure 1. These three dependencies are addressed by coordination mechanisms and these mechanisms are arranged into categories called coordination strategy components with subcategories each, as shown in Table 3. Following sections will describe the coordination strategy components working as coordination mechanisms by Strode et al. (2012) and the taxonomy of dependencies by Strode (2016).

Coordination Strategy Components

Coordination strategy is defined as a group of coordination mechanisms that manage dependencies in a situation and the strategy has three components: synchronization, structure, and boundary spanning (Strode et al., 2012). Table 3 provides the definitions for the coordination strategy components and is also defined as a theory of coordination in agile software development projects.

Table 3: Definitions of coordination strategy components (Strode et al., 2012)

Distinct component	Component	Definition
Synchronization	Synchronization activity	Activities performed by all team members simultaneously that promote a common understanding of the task, process, and or expertise of other team members.
	Synchronization artefact	An artefact generated during synchronization activities. The nature of the artefact may be visible to the whole team at a glance or largely invisible but available. An artefact can be physical or virtual, temporary or permanent.
Structure	Proximity	This is the physical closeness of individual team members. Adjacent desks provide the highest level of proximity.
	Availability	Team members are continually present and able to respond to requests for assistance or information.
	Substitutability	Team members are able to perform the work of another to maintain time schedules.
Boundary spanning	Boundary spanning activity	Activities (team or individual) performed to elicit assistance or information from some unit or organization external to the project.
	Boundary spanning artefact	An artefact produced to enable coordination beyond the team and project boundaries. The nature of the artefact may be visible to the whole team at a glance or largely invisible but available. An artefact can be physical or virtual, temporary or permanent.
	Coordinator role	A role taken by a project team member specifically to support interaction with people who are not part of the project team but who provide resources or information to the project.

Dependency Taxonomy

In coordination mechanisms, there are necessary to address dependencies in a situation. This is one of the central principles of coordination (Strode, 2016). Dependencies are proposed by Crowston and Osborn (1998), with a definition of dependency: "a dependency occurs when the progress of one action relies upon the timely output of a previous action or on the presence of a specific thing, where a thing can be an artefact, a person, or a piece of information. When dependencies occur in a development project, they can be managed well, poorly, or not at all" (Strode, 2016, p. 24). A taxonomy done by Strode (2016) was

developed to organize knowledge about dependencies. This taxonomy is built on the theory of coordination, coordination mechanisms and dependencies by Strode (2012). Taxonomy in information systems is described with theories with respect to the manner in which four central goals are addressed: analysis, explanation, prediction, and prescription (Gregor, 2006). Taxonomies are useful when little is known about a topic. Then concepts and constructs need to be identified (Strode, 2016).

The taxonomy including *knowledge*, *process*, and *resource* dependencies. *Knowledge* dependencies consist of requirements, task allocation, historical, and expertise dependencies; *process* dependencies include activity and business process dependencies; and *resource* dependencies include entity, and technical dependencies (Strode, 2016). Figure 1 give an overview of this dependencies and a description of each is described in Table 11 in Chapter 5.

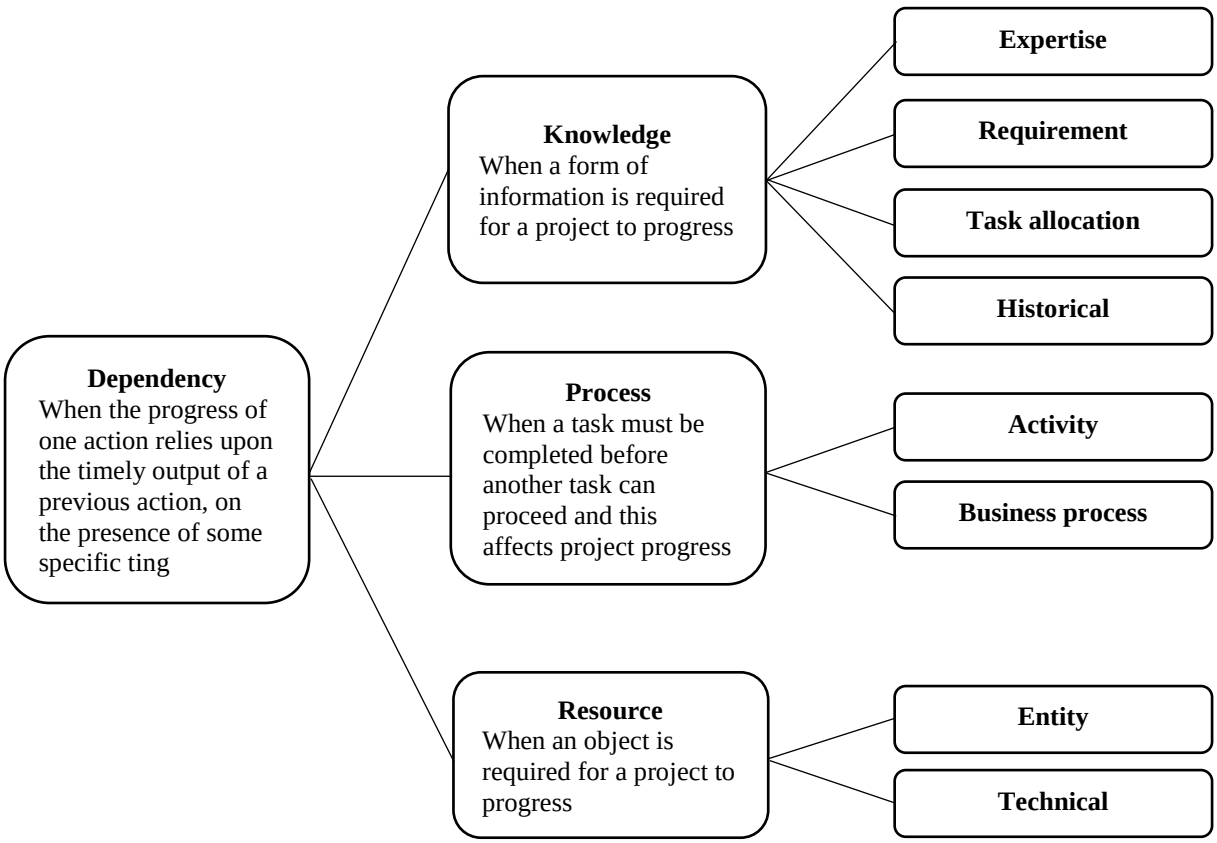

Figure 1: Schematic of the dependency taxonomy (Strode, 2016)

2.4 Autonomous Cross-Functional Teams

A team is a group of people with a high degree of interdependence and is aimed at achieving a goal or completing a task (Parker, 2003). Team members confirm the goal and confirm that the only way to reach the goal is to work together. There are groups that are not teams, this is groups with the same goal. The best-known types of teams are functional teams, self-directed teams and cross-functional teams (Parker, 2003). As Parker states in his book about a world of business are changing: "Individualism is out, teamwork is in. Power is out, empowerment is in. Hierarchical organizations are out, replaced by network organizations, adaptive organizations informational organizations and horizontal organizations" (Parker, 2003, p. 1). In the middle of this statements sit cross-functional teams, to changing business needs composed of experts ready to move quickly and flexibly (Parker, 2003).

Cross-functional teams are structured as working groups created to make decisions lower in an organizations hierarchy (Denison et al., 1996). Cross-functional teams differ in important ways and share many characteristics with conventional teams. First, cross-functional teams are "usually representative groups in which each member has a competing social identity and obligation to another submit of the organization" (Denison et al., 1996, p. 1005). Second, the team "are often temporary task teams experiencing abundant pressure and conflict, so the early development of stable and effective group processes is critical to their success" (Denison et al., 1996, pp. 1005-1006). Third, the team "typically confront a different set of performance expectations than conventional work teams and are often expected to reduce cycle time, create knowledge and disseminate organizational learning" (Denison et al., 1996, p. 1006).

Furthermore, cross-functional teams work best in markets that work with environments like agile development. The reason for this is because of those teams are most effective in companies working in a rapidly changing market, such as IT, telecommunications, the pharmaceutical industry, and other industries that value adaptability, speed and a high focus on meet customer needs in the market (Parker, 2003). Cross-functional teams bring six important advantages to organizations that successfully implement and manage them as shown in Table 4:

Table 4: Important advantages in cross-functional teams (Parker, 2003, pp. 12-13)

Advantages	**Description**
Speed	Cross-functional teams reduce the time it takes to get things done, especially in the product development process.
Complexity	Cross-functional teams improve an organizations ability to solve complex problems.
Customer focus	Cross-functional teams focus the organizations resources on satisfying the customers need.
Creativity	By bringing together people with a variety of experiences and backgrounds, cross-functional teams increase the creative capacity of an organization.
Organization learning	Members of cross-functional teams are more easily able to develop new technical and professional skills, learn more about other disciplines, and learn how to work with people who have different team-player styles and cultural backgrounds than those who do not participate in cross-functional teams.
Single point of contact	The cross-functional teams promote a more effective cross-team effort by identifying one place to go for information and for decisions about a project or customer.

Moreover, when it comes to cross-functional teams, it exists autonomous teams as well. In the literature, autonomous teams are defined as "self-managed teams, empowered work groups, or self-directed work teams" (Janz, Wetherbe, Davis, & Noe, 1997, p. 43). A recent study by Patanakul, Chen, and Lynn (2012) involved a case study of autonomous teams and new product development. They described autonomous teams as separate from the mainstream organization, having its own members handling development, manufacturing, and marketing. Autonomous teams "is a team whose members typically are dedicated and collocated with a project leader who is a senior manager in the organization" (Patanakul et al., 2012, p. 736). Autonomous teams should be appropriate for a project with a high degree of technology innovation. Lastly, such teams allow rich frequent communication, decentralized decision making, and high levels of cross-functional integration (Patanakul et al., 2012).

2.4.1 Teams vs. Groups

"Despite what we call team, not all "teams" are teams. Some so-called teams are simply groups masquerading as teams because in today's world it is important to be on something called team" (Parker, 2003, p. 1). Other statements about groups are done by Katzenbach and Smith (2005) why describe that "the best working groups come together to share information, perspectives and insights; to make decisions that help each person do his or her job better; and to reinforce individual performance standards". It is possible to compare a working group and a team. Table 5 shows this comparison, and is based on Katzenbach and Smith (2005, p. 4):

Table 5: Working group vs. a team

Working group	**Team**
Strong, clearly focused leader	Shared leadership roles
Individual accountability	Individual and mutual accountability
The group's purpose is the same as the broader organizational mission	Specific team purpose that the team itself delivers
Individual work products	Collective work products
Runs efficient meetings	Encourages open-ended discussion and active problem-solving meetings
Measures its effectiveness indirectly by its influence on others (such as financial performance of the business)	Measures performance directly by assessing collective work products
Discusses, decides, and delegates	Discusses, decides, and does real work together

2.5 DevOps

DevOps is an interesting concept in the web domain. DevOps has two core principles highlight collaboration between development and operations in software and it is a clipped compound of this words. The concept uses agile principles to manage deployment environments and their configurations (Lwakatare et al., 2016). The DevOps concept is also a collaboration between automation and the use of new tools and technologies (Wiedemann, 2018). DevOps is a quite new phenomenon in software engineering and "the main goal is to shorten feedback loops and the development cycle through collaboration, automation and frequent software releases" (Lwakatare et al., 2016).

So far, however, there has been little discussion about this gap between the collaboration between departments in companies. In most company's development and operations exist as separate functions, therefore the collaboration in DevOps seeks to bridge the silos of software development and operations functions (Lwakatare et al., 2016). Figure 2 shows the flow of the DevOps concept and this collaboration is essential when new software features are developed and released to the customer frequently and quickly on a continuous basis.

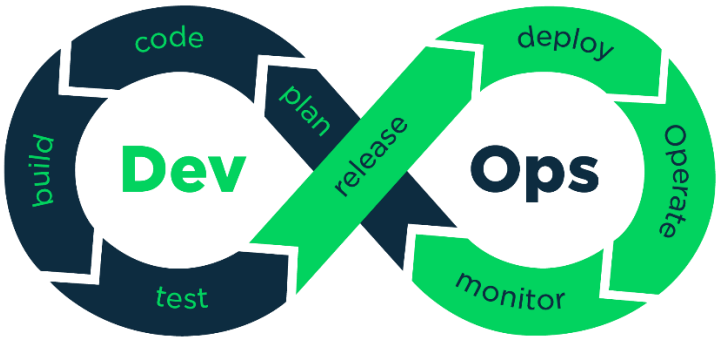

Figure 2: The flow in the DevOps concept (Kornilova, 2017)

The concept of DevOps can cause changes in the internal IT functions when the implementation of it is done. The changes are reflected in new processes, structures, and governance mechanisms. Wiedemann mentions that "some organizations have already started to adapt their IT functions. Incumbent companies have to rethink their IT governance mechanisms within dynamic and agile environments" (Wiedemann, 2018). A small-scale study by Fitzgerald and Stol (2014) reaches different conclusions about trends and challenges in software engineering. They mention that departments should implement cross-functional teams for fast delivery of new software features, innovations and quick handling of problems. One service should be conducted by a single team for all necessary activities for the software delivery cycle. Furthermore, the DevOps broadens the agile approach by applying continuous integration, defined as a process that is triggered automatically and includes interconnected stages. E.g. testing, release package development and code validation (Fitzgerald & Stol, 2014).

3 Research Method and Design

The previous chapters have motivated the need for this thesis, presented research questions, relevant case studies, and theory. To answer the research questions, this chapter present and provide the research method and design used in this thesis and technique for data analysis. First, there will be a brief content of qualitative research. Then data collection and data analysis will be introduced. Lastly, a brief introduction to validity will be presented.

3.1 Qualitative Research

When investigating the research questions, I started to develop the research design. Research methods may be qualitative or quantitative (Merriam, 1998; Patton, 1990; Yin, 2002). The differences between these methods can be discussed in different ways. One statement is Yin (2002), he argues against those who make distinctions between the methods due to the irreconcilable philosophical disparities: "regardless of whether one favors qualitative or quantitative research, there is a strong and essential common ground between the two" (Yin, 2002, p. 15). He does not distinguish between quantitative and qualitative case study methods. He attends to the commonalities of the two research methods and pragmatically foregrounds the common tools which can be functional and instrumental in the design and methods of case study he suggests.

However, since my research questions involve coordination mechanisms between peoples decided by peoples, qualitative methods seemed most to fit the research design. In the qualitative research methods, there are three traditional strategies, namely case study, ethnographic study and grounded theory study (Gerring, 2007), where the conducted strategy in this thesis is a case study. Furthermore, in case studies there are four ways to collect data: interviews, observation, written documents, and audio and video material (Johannessen, Tufte, & Christoffersen, 2010).

3.1.1 Case Study

A case study was conducted to investigate the research questions of how the effects of coordination in autonomous DevOps teams can be adapted to large-scale agile development in software engineering. A case study in general, the preferred strategy is to answer "how" or "why" questions (Yin, 2002, p. 1). In this case study, it is highly appropriate at answering

the research questions regarding "how" different cross-functional teams coordinate themselves, and "why" the coordination mechanism found in a research period is used.

The chosen type of data collection is participant observation. Gerring (2007) states Platt (1992), where she notes that "much case study theorizing has been conceptually confused because too many different themes have been packed into the idea case study" (Gerring, 2007, p. 18). Furthermore, Gerring (2007, p. 20) writes: "a case study may be understood as the intensive study of a single case where the purpose of that study is – at least in part to shed light on a larger class of cases". Of these statements, there is no clear rule for how to conduct a case study or what it is.

Moreover, in this research, a team of analysis is a group of people, composed of a so-called autonomous DevOps teams. However, in addition to the teams of analysis later called team Venus, team Mars, team Jupiter, team Pluto, team Customer and team Earth. There was another team, later called team Saturn, that had recently occurred into a new team with a mix of members from team Venus, team leads, Product Owners and other outsourced developers from other consultant houses. I, therefore, collected data from all the teams and other units in the organization to better understand the large-scale program.

Figure 3 shows the context of this study which both the development unit (Dev) and the operation unit (Ops) were working in the same project area with different teams. Moreover, where the teams working in the same project area before and after the scaling process.

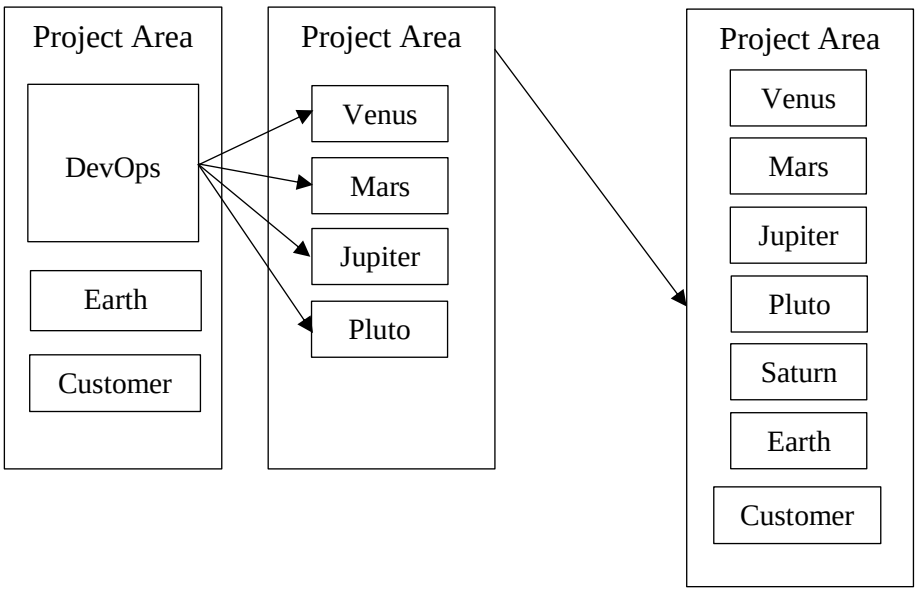

Figure 3: The context of the reorganized structure after the scaling process

3.1.2 Research Sites

The organization under study is a Norwegian department, called Knowit Objectnet AS at a Swedish consultancy company named Knowit AB. The company creates customer value in a world of accelerating digitalization and offering international solutions in design and communication, management consulting and IT. The selected company have offices in 14 locations in Sweden, five in Norway and one each in Denmark, Finland, and Germany. The company had a net sale on 2,426.2 SEK million in 2016 and is listed on the Nordic Exchange in Stockholm. The company is divided into three divisions: Experience, Insight, and Solutions. The company has around 2000 employees all over, and the participant observations were conducted at the Solutions department at the head office in Oslo, Norway, with around 40 persons involved in the project. This department helps companies and organizations to develop their activities through a range of IT solutions. The largest customer in this department is in public sector. This case study is therefore about a case in the public sector, namely the digitization process for a municipality. The company was chosen because it is part of a research group on agile methods and autonomous team for global software development.

However, the research study participants in the early phase were through a snowball sampling technique (Patton, 1990). As stated by Patton (1990, p. 177) this technique "is an approach for locating information-rich key informants or critical cases. The process begins by asking well-situated people: "Who knows a lot about…? Who should I talk to?"". The technique was conducted by contacting people in a higher section of the department at the consultancy company. After getting contact with persons, a person gave access to a range of project teams and stakeholder with different perspectives. Participants from a big project that matched the agile large-scale approaches were selected after a discussion between the supervisor, the consultancy company, and the stakeholder. Later in the study, intensity sampling was used to obtain greater richness by targeting observation of different responsibilities in the same project. The intensity sampling technique is stated by Patton (1990, p. 182) as: "Information-rich cases that manifest the phenomenon intensely, but not extremely, such as good students/ poor students, above average/below average".

Moreover, the project under study, develop solutions in the public sectors. The goal in the public sector is to streamline operations and to simplify communication with citizens. Examples of solutions they provide are web solutions, mobile solutions, document handling

solutions and business systems. 1th of January 2018, there was a reorganized shift between the supplier and the customer. The customer introduced three more suppliers in the same project. Therefore, customer reorganized the coordination mechanisms by adapting to divide the tasks into multiple suppliers.

To summarize, the research site in the early phase of the study was selected to provide replication using the snowball technique sampling. Lastly, in the later phase, the intensity sampling was used to enhance both depth and richness. Hence increasing data reliability through participant observation.

3.2 Data Collection

The data may be collected in naturalistic analysis approaches. The qualitative data are the primary focus of naturalistic inquiry; there controlled experimental designs predominantly aim for statistical analyses of qualitative data (Patton, 2002). "Qualitative data describe. They take us, as readers, into the time and place of the observation so that we know what it was like to have been there" (Patton, 2002, p. 47).

Another central activity of qualitative analysis is fieldwork. Likewise, a qualitative analysis is described by Patton (2002, p. 48) as "into the real world of programs organizations, neighborhoods, street corners and getting close enough to the people and circumstances there to capture what is happening." Furthermore, fieldwork is described as: "going into the field." This means having personal contact with people under study in their own environments (Patton, 2002).

The data that was collected in this master thesis is sources based on evidence of participant observation. In Table 6 the four strategies of data collection and fieldwork put forth by (Patton, 2002, p. 40) were followed when collecting the data. This was a part of the themes of qualitative analysis.

Table 6: Four strategies of data collection and fieldwork

Strategies	**My approach**
Qualitative data	I used participant observation that yield detailed, thick description; analysis in depth.
Personal experience and engagement	Under the study, I was in direct contact with the people and situation. And for me personally, experiences and insights were an important part of the analysis and critical to understanding the phenomenon.
Empathic neutrality and mindfulness	In the observation under the study, the mindfulness was fully presented.
Dynamic systems	Mindful of and attentive to system and situation dynamics was given. Attention to the process was given by change as ongoing whether the focus was on an individual and an organization.

3.2.1 Observation

Through this study, I was able to observe one department in the organization to get insight into their way of working. To answer the research questions, the observation method was selected. According to Johannessen et al. (2010) observations are detailed descriptions of human activities, behavior or actions as well as interpersonal interaction and organizational processes. "The observation is best suited as a method when the problem is linked to a limited geographical area" (Johannessen et al., 2010, p. 120). Since the organization under study was the access to the field, the data produced greatly influenced the validity of the knowledge. In section 3.4 the principles around validity will be presented.

Moreover, when document the observations it is possible to separate structured and unstructured observations (Johannessen et al., 2010). Structured observation means that the researcher operates with a form that contains predetermined categories that determine what should be observed and recorded (Johannessen et al., 2010). Based on this the observations were guided by an observation protocol based on Spradley (1980) and Stray et al. (2016), see Appendix A. A structured observation was chosen, during the observations, notes were written with general information, such as a number of attendees, content, start and end time. Morover, the observations lasted from November 2017, to April 2018.

The primary data used in the study were from observation, and it was observed types of meetings as shown in Table 7. The visits enabled observation of working practices and

workplace environments in the organization. 40 coordination meetings were observed for all the seven teams in the project, as well as 18 entire working days in the organization's open work area where the teams were situated in order to observe them in their everyday work. Various informal, sometimes offsite, discussions with executives, project management, and development team members were conducted during the lunch break or in a natural way in the open work area.

Table 7: An overview of the meetings observed

Observations	**Total**	**Team Observed**
Daily Stand-up	12	3 team Jupiter, 3 team Mars, 2 team Saturn, 4 team Venus
Demo meetings	6	Participants from all 7 teams
Sprint meetings	2	Participants from team Venus
Scrum of scrums	5	Participants from all 7 teams
Project meetings	7	Participants from team Earth and the customer
Workshop	3	Participants from Jupiter, Pluto, Mars, Saturn and Venus
Team lead meetings	2	Participants from Jupiter, Pluto, Mars, Saturn and Venus
Futurespective	1	Participants from team Pluto
Other meetings	2	Participants from team Earth and the customer
Sum:	**40**	

3.3 Data Analysis

To make sense of the data, I analyzed the data. My first step in data analysis was to prepare a summary and a reflection paper of each note from the meetings and other observed material. In total I analyzed over 60 pages from the observed meeting notes. The reflection paper included details of the organization under study, the project, the teams, the meetings, the roles, and other coordination observations.

My analysis built on theories presented in Chapter 2 on agile software development and large-scale agile development, coordination, and teamwork. The taxonomy of dependencies and coordination mechanisms proposed by Strode (2016) was used to getting an overview of the field of dependencies and coordination mechanisms. The results in Chapter 5 are organized according to the dependency taxonomy by Strode (2016), and an example of the coding process performed in my study is shown in Figure 4.

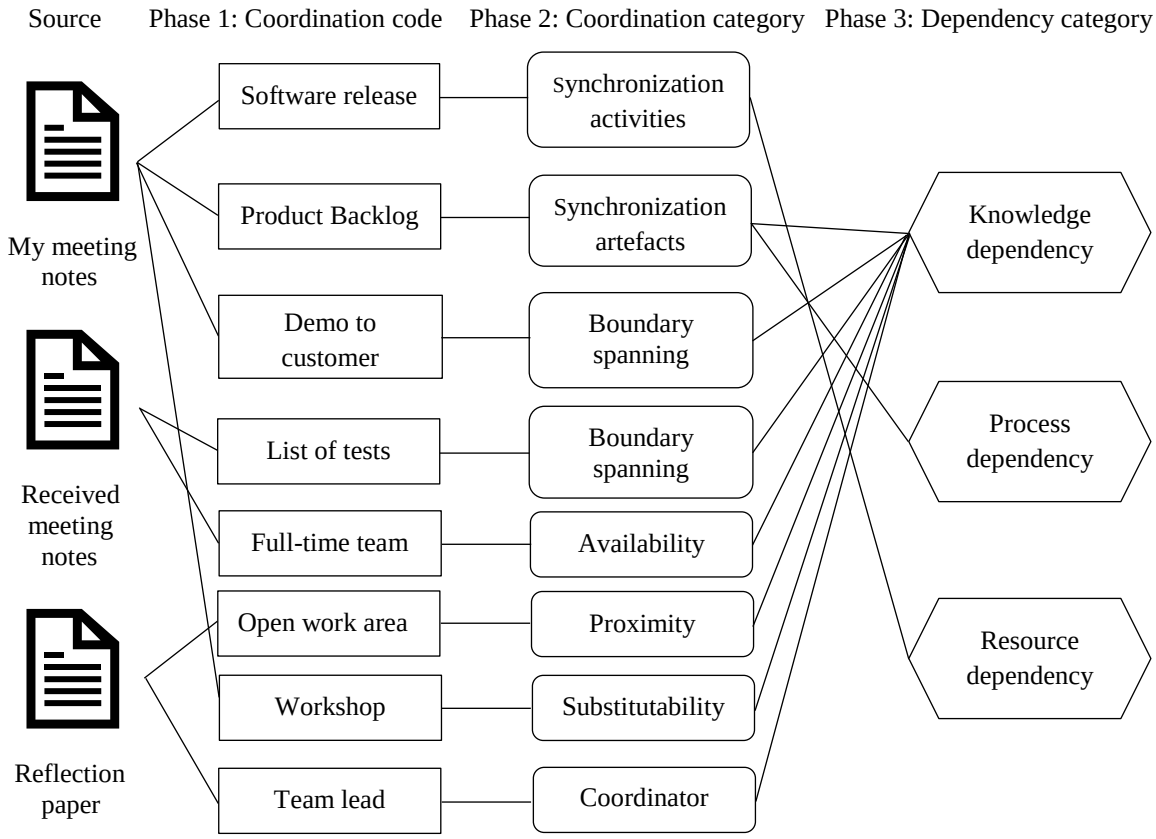

Figure 4: An example of the coding process in my study

To conduct the data analysis, all the data sources were uploaded into a software program tool called QSR NVivo. The software is designed for analyzing qualitative data coding. A general inductive coding technique was followed (Miles & Huberman, 1994), beginning with starter codes involved identifying data items that shared an ordinary meaning. The data items were given a descriptive name, called a code, and each code was defined uniquely. This analyses aimed to identify dependencies and their associated coordination mechanisms, so the coding approach was guided by Crowston and Osborn (1998).

Figure 4 is more detailed and described here: In the first phase, coordination mechanisms were identified from the sources meeting notes and the reflection paper. In the second phase, the coordination mechanisms were mapped into a coordination mechanism category. In the third phase, dependencies were grouped, such as knowledge, process and resource dependencies. Then, the coordination mechanisms from the first phase were mapped to one or more dependencies.

3.4 Validity

The previous chapters have described the qualitative research method with a case study and research sites, data collection and the data analysis. A key question in research is then how good or relevant data represent the phenomenon. The research literature uses the term *validity*. A distinction is made between different forms of validity, including *construct validity, internal validity* and *external validity* (Johannessen et al., 2010). The approach to increasing the validities in this case study is outlined in the implications for theory, section 6.4 in Chapter 6.

Construct Validity

Construct validity is concerned about the relation between the general investigated phenomenon and the specific data. We have to ask ourselves (Johannessen et al., 2010): Are the data good, valid representations of the general phenomenon? Validity must not be perceived as absolute if data is valid or not, but it is a quality requirement that can be virtually satisfied. Construct validity is a typical measurement phenomenon. It is a matter of whether there is a match between the general phenomenon to be investigated and the measurement (Johannessen et al., 2010).

Internal Validity

The internal validity is relevant when an experiment has carried out that a proven relation between two variables concerns a possible causal connection (Johannessen et al., 2010).

External Validity

According to Johannessen et al. (2010, p. 231) "the research cannot be limited to pure data collection. The information must be systematized and analyzed. The analysis involves coded information". In external validity increases the use of theory, and the external validity develops theories, concepts, and interpretations that illustrate the phenomenon of your study (Johannessen et al., 2010).

4 Research Context

This chapter will explain the research context of this research. First, a brief description of the organization and project case under study will be presented. Then, the teams will be introduced with a detailed description and seating map. Lastly, an overview of the roles will be presented.

4.1 The Organization and Project Case

Since the chosen department in this case study developing IT solutions for the customer, the research subjects under study are seven teams. The seven teams under study are called team Jupiter, team Pluto, team Mars, team Saturn, team Venus, team Earth, and team Customer. This altogether makes it to a large-scale agile program. The five DevOps teams Jupiter, Pluto, Mars, Saturn, and Venus develop and operate products from the conceptual phase to the finished solution. The main role of team Earth is to support the DevOps teams with User Experience (UX) designers, different architects, and Product Owners. One team standing outside, team Pluto, they are outsourced and located at one of the customers department. Their Product Owner is from one of the customers department. Moreover, the team members from the customer have a central part and are involved with a project manager in front, a test lead, architects, designers, a security manager and representatives from different departments. Express in other words; they are a part of this large-scale program with own resources. The next section will detail this unit by illustrated overviews.

4.1.1 The Large-Scale Agile Program

The large-scale project under study was studied in two different periods. The first period, autumn 2017 was the period with just one supplier. Knowit was the only company that developed the solutions and had full responsibility for the DevOps teams. All the participants in this DevOps teams were a project manager, developers and team leads from their own house. Only the Product Owners was participants from team Earth who was related to the DevOps teams but was still standing outside. Figure 14, in section 4.3 gives a more detailed overview on this.

By an organizational map, Figure 5, illustrates the large-scale agile program before the scaling process.

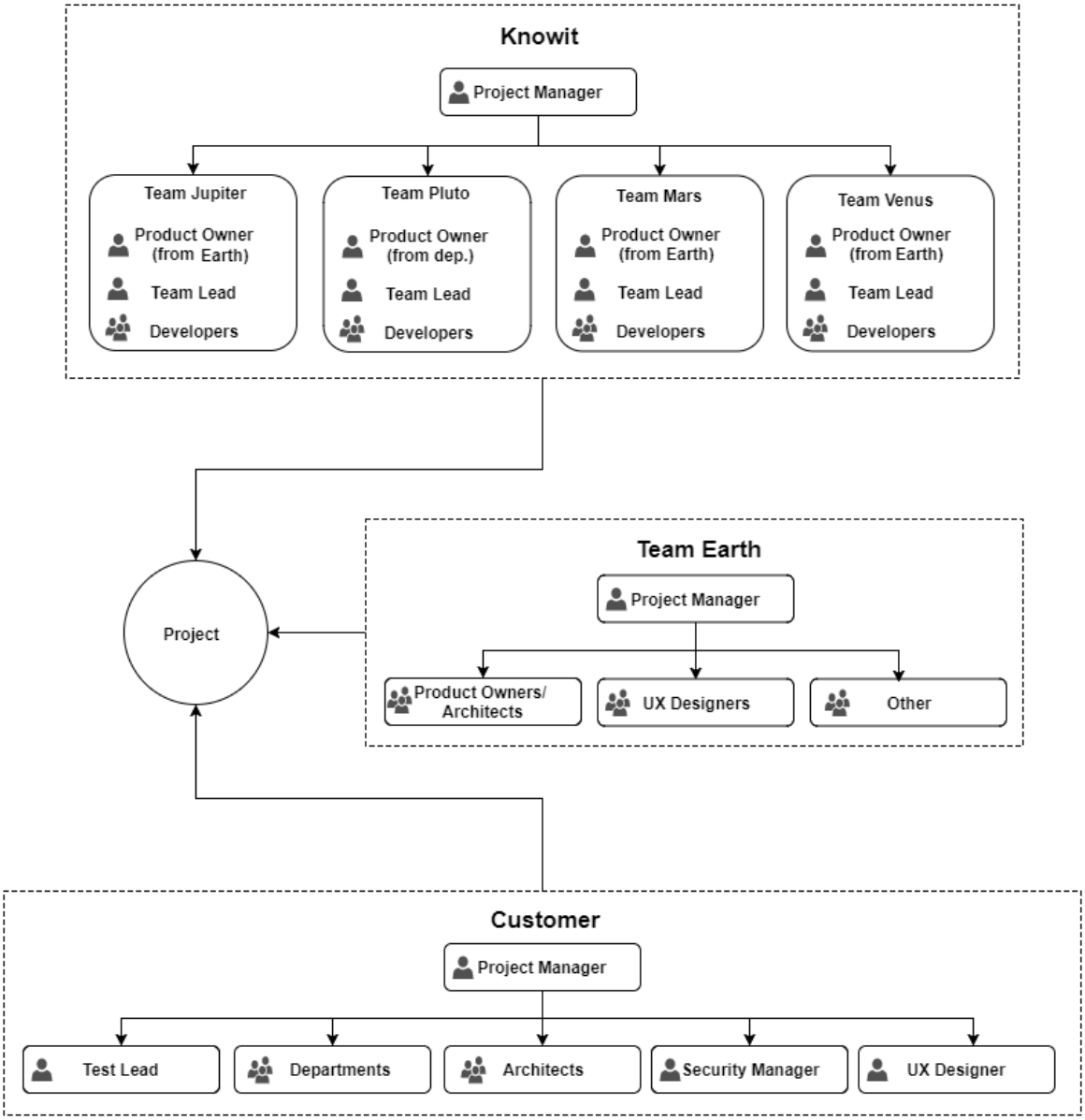

Figure 5: The large-scale agile program before the scaling process

Moreover, after 1st of January 2018, there was a scaling process for the customer. Three new suppliers were retrieved, with the names company B, C and D. A new structure of the large-scale agile program was reorganized as showed in a timeline in Figure 6:

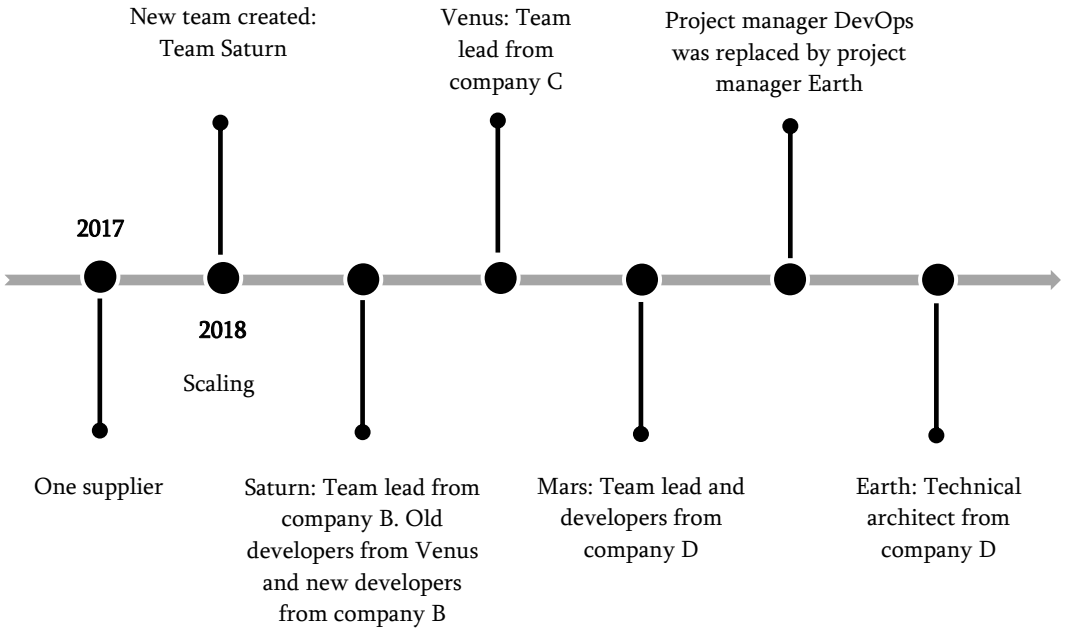

Figure 6: A timeline of the large-scale program

In this large-scale agile program, the scaling process was a request from the customer and this resulted in a major change of team size, stakeholders, and reorganizations at the management level. An organizational map Figure 7 illustrates the large-scale agile program after the scaling process.

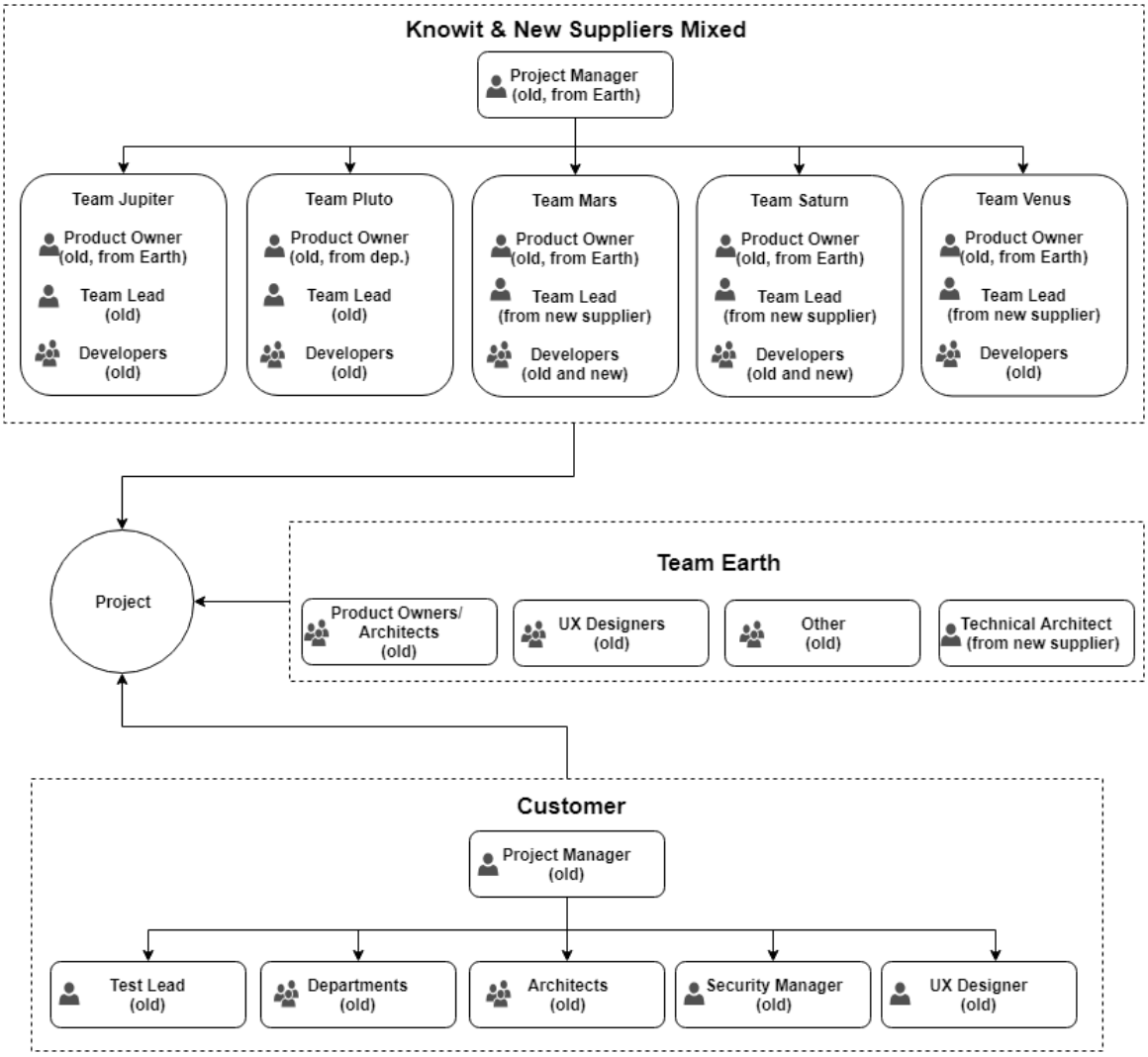

Figure 7: The large-scale program after the scaling process

4.1.2 Work Area

The large-scale agile program was situated at Knowits offices, and the open work area was spread over two floors. Figure 8 shows the open work area and the status of the teams on the 4th floor. All the UX designers were placed in a small room at the end of the area. Team Earth with Product Owners and the project manager for the DevOps teams were set in an open area beside the UX designers working room. Project manager, test lead, and architects from the customer were also placed in this area with the team Earth. Team Saturn was placed nearby team Venus and team Mars with a meeting room in between them. The participants from team Mars in the 4th-floor area were some developers and their Product Owner from team Earth. The rest of team Mars was located on the 5th floor, see Figure 9.

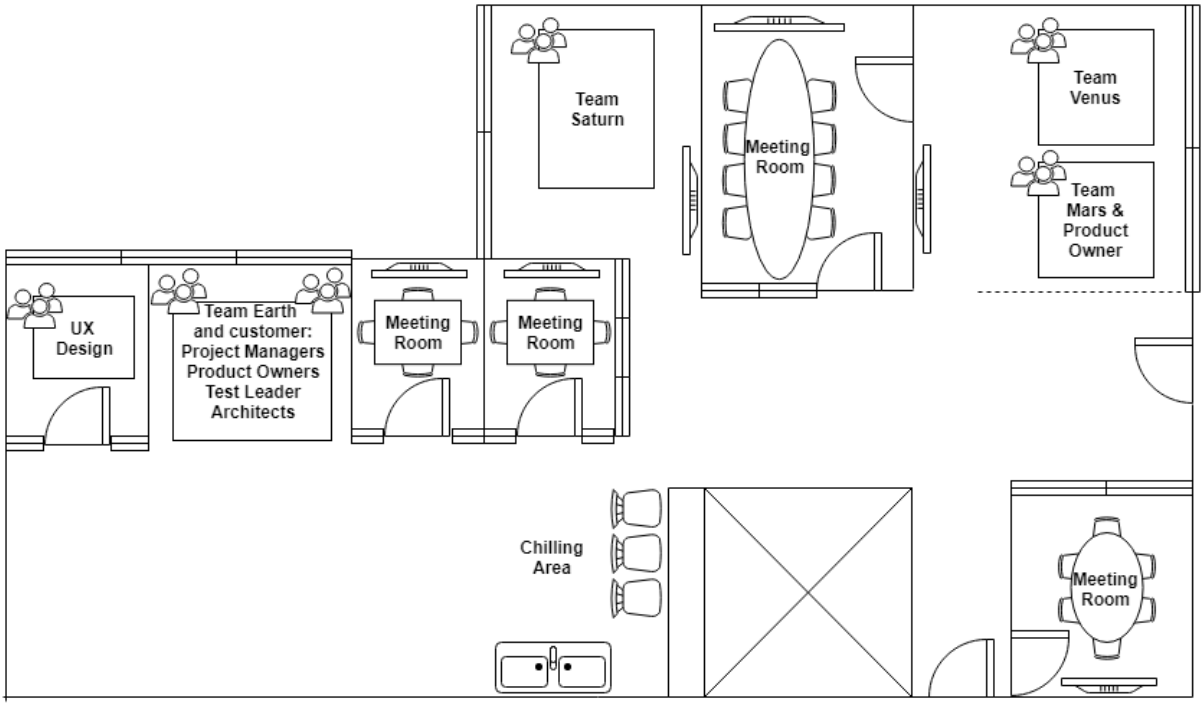

Figure 8: Open work area where the program was situated on the 4th floor under this study

Figure 9 shows the open work area and the status of the teams on the 5th floor. The rest of the team Mars were placed in an own working room. Team Jupiter was situated in the same area as team Mars with a wall between them. Team Pluto was off-site and located at one of the customers departments a minute from the head offices. But sometimes some of the developers were placed on the 5th floor. Under this study, this did not happen often. Team Pluto was therefore not observed as much as the rest of the teams in this large-scale agile program.

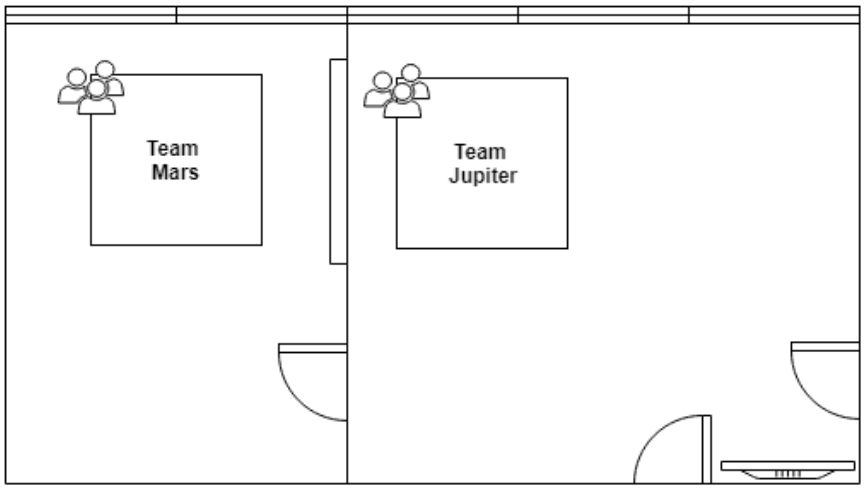

Figure 9: Open work area where the program was situated on the 5th floor under this study

As we can see of this part in the study, the DevOps teams on the 4th floor are placed on the same level as the Product Owners, the product managers, the UX designers, and the architects.

4.2 The Investigated Teams

Table 8: Team overview

Team	Members	Located	Responsibility
Jupiter	7	5th floor	Developing to the commercial department
Pluto	5	Off-site	Developing to one of the customers department
Mars	8	5th and 4th floor	"Min side", developed by Difi
Saturn	5	4th floor	Data-Driven Guided Dialogue (DVD)
Venus	9/4	4th floor	Common components and API
Customer	6	4th floor	The customer, project is up and running
Earth	8	4th floor	Hired resources, Product Owners, UX Designers
Total members	**48**		

Table 8 gives an overview of the teams, a number of the members, where they are located, and a description of their responsibilities. In the text sections, the teams will be more described in detail.

4.2.1 Team Jupiter

Team, Jupiter has the responsibility for developing solutions to the commercial department. The team is a mix of junior and senior developers, and team Jupiter took place in the early phase of the project with a team lead with a lot of experience over several years.

The team consists of seven representatives, one team lead and six developers. However, some of the team members possess multiple roles. Of the six developers, there is one back-end developer and five front-end developers. From the team Earth, there is one Project Owner and one technical architect who support the team. There is not a defined test role in this team. Of the six developers, they are responsible for keeping the testing up and running by unit testing internally within the team. Sometimes testing material is sent to the test lead before deploying to the customer. Team lead has the overall responsibility for the products being developed.

The developers are placed in open-plan offices with the team lead on the 5th floor. The front-end and back-end developers are sitting mixed with the team lead at one of the ends, see Figure 10. The Product Owner and the technical architect are not placed with the team; they are located on the 4th floor.

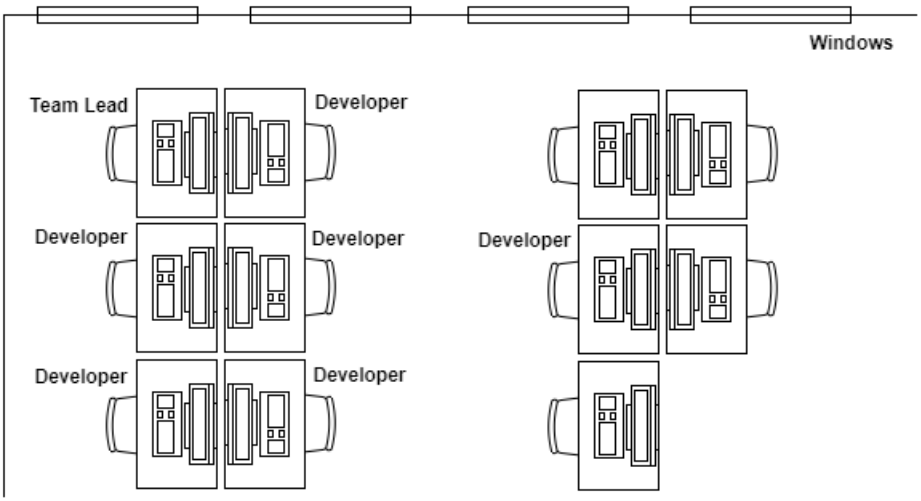

Figure 10: Team Jupiter's seating arrangement

4.2.2 Team Pluto

Team Pluto has the responsibility for developing solutions to one of the customers department. Team Pluto is a team with a team lead with much experience who joined the project in an early phase. The team is a mix of junior and senior developers, and this team is the off-site team.

The team consists of five representatives, one team lead and four developers. Of the four developers there is one who is working as front-end and back-end developer, one developer, is working with only front-end and two developers with the only back-end. The Product Owner for this team is a person from one of the customers department, and from team Earth, the technical architect supports the team with the architectural part.

While this team working off-site, the team is still on-site. The team are located just a minute from the main offices and attends the scheduled meetings by a short walk from their locations.

4.2.3 Team Mars

Team Mars has the responsibility for "Min side". This is a solution that offers services and information that pertains the users by a login portal developed by the Agency for Public Management and eGovernment (Difi). Before the reorganization one of the team leaders also joined the project in an early phase with over 20 years of experience in the company.

The team consists of eight representatives; one team lead who working 50% in this case and another member who is working 50% as a team lead and 50% as a developer. This team lead work as a front-end developer. Of the six developers, there are some of those who have optional roles with a combination of both back-end and front-end development. Apart from these, there are clear roles between who is front-end and back-end developers. Some developers come from the new suppliers. The Product Owner and the technical architect for this team are representatives from the team Earth. Furthermore, neither in this team, there is not a defined test role. The team members are responsible for keeping the testing up and running by unit testing and manual tests.

Team Mars is also placed on the 5th floor in open-plan offices in a closed room just a few meters from team Jupiter, see Figure 11. All the team members on this floor are located next to each other, but two of the developers located with the Product Owner who is also

working as a solution architect is placed next to team Venus on the 4th floor. This map is shown in Figure 14.

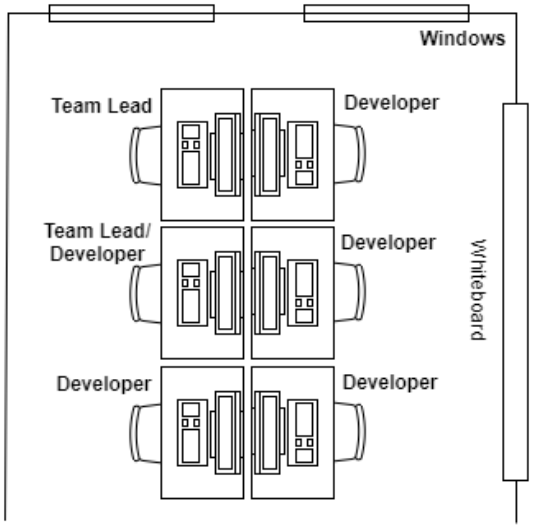

Figure 11: Team Mars seating arrangement

4.2.4 Team Saturn

Team Saturn was established in January 2018 after team Venus was divided. This team has the responsibility for the Data-Driven Guided Dialogue (DVD)-framework. Team Saturn is, therefore, one of the latest team in this large-scale program with a new team lead which comes from one of the new suppliers.

The team consists of five representatives, one team lead, and four developers. The team lead is responsible for the product delivery and sometimes working also as a developer. Of the four developers, there is two front-end and two back-end developers. These developers specialize in these areas, and the team members are a mix of old developers and new developers from the new suppliers. The old team members were part of a former major team Venus after the team divided in the scaling process. Team Saturn's main goal is to share equal skills among the team members on the product delivered to the customer. The Product Owner and the technical architect for this team are representatives from team Earth. Testing is done via unit testing and automated testing. Final tests before the product launch are done via the test lead.

Team Saturn is located at team Venus old open-plan offices on the 4th floor, see Figure 12. All the developers are located next to each other with the team lead placed by himself with

empty spaces between them. The empty spaces in the open-plan offices take place because former team Venus had more team members than today's team Saturn.

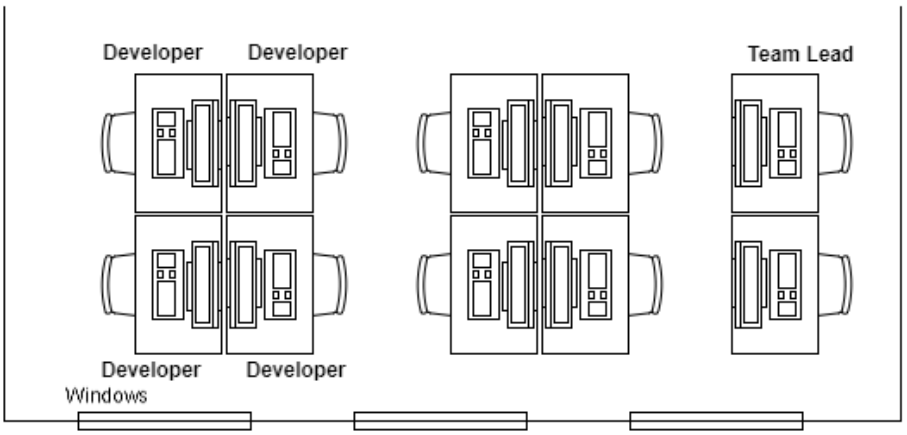

Figure 12: Team Saturn's seating arrangement

4.2.5 Team Venus

Team Venus has the responsibility for all common components and API components in the project and working with typically back-end-oriented tasks. Team Venus is an old team in this large-scale program, but after the scaling process, the team was reorganized with a new team lead and some new developers from the new suppliers. Before the scaling process, team Venus was the main team in the program and by far the greatest team.

Before the scaling process team Venus consist of nine team members and after the reorganization the team was divided into just four team members, one team lead and three developers. Some of the developers remained in the team, and other developers joined team Saturn. Of the three developers all of them is typically back-end developers since the team focuses on back-end-oriented solutions, but with different programming skills. The Product Owner and the technical architect for team Venus is also representatives from team Earth. Test lead is being contacted before deploying to the customer. Otherwise, testing will be done by unit testing, automated testing and functional testing by the developers.

Before the scaling process, team Venus was located on the 4th floor next to team Earth and the team from the customer. However, the developers were placed in open-plan offices with the project manager from Knowit next to the team lead, see Figure 13. Since this team was the biggest and greatest team, it was natural to place the project manager next to team Venus.

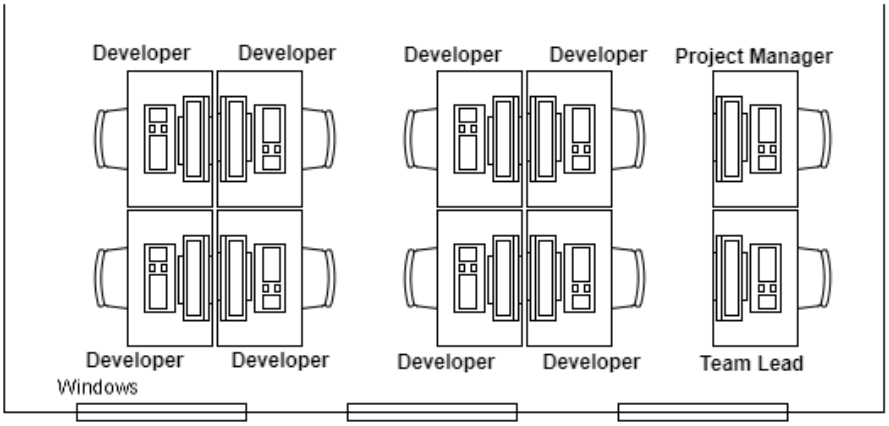

Figure 13: Team Venus seating arrangement before the scaling process

After the scaling process, the team were divided and was placed in a new smaller open area next to some team members from team Mars as shown in Figure 14. The project manager for the DevOps teams leave the project, and the managers place in the open-plan office remained standing empty after team Saturn took over the location, see Figure 12.

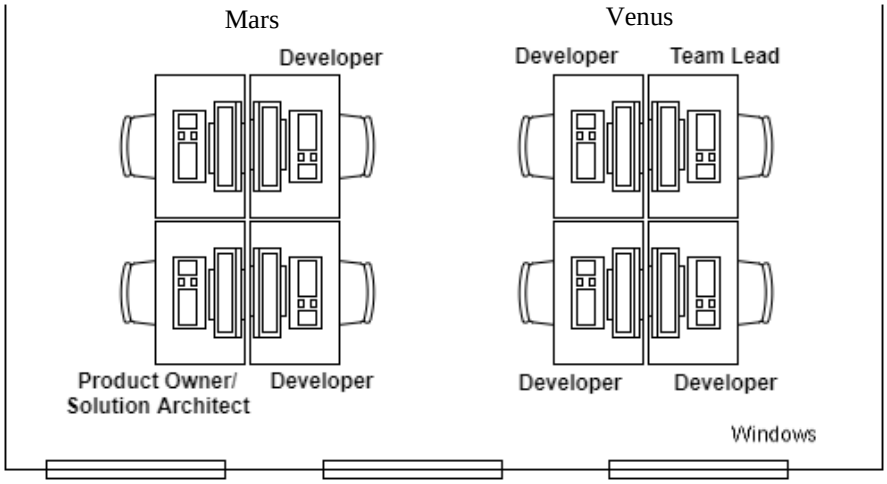

Figure 14: Team Venus and team Mars seating arrangement after the scaling process

4.2.6 Team Customer

The customer are previously mentioned as the customer in this project. They are also involved with their resources from their IT department and other departments. With a project manager in the lead, the manager has the main responsibility for the project is up and running. Next to the manager, we can find one test lead, architects, a security manager and one representative UX designer who makes them to a separate team in this large-scale program. The team is placed next to team Earth in the same open-plan offices and they practiced free seating. Figure 8 shows their location in the open-plan office.

4.2.7 Team Earth

In addition to the DevOps teams and the team from the customer above, the customer hire consultants from other technology companies. These participants are product owners who also act as different architects. Furthermore, UX designers and other types of architects are also hired. Before the scaling process, the project manager for team Earth was hired from a management consultant company. After the scaling process and reorganizations in the project, the manager from team Earth joined the DevOps teams and acts now as the project manager for the DevOps teams. After the reorganizations, a technical architect joined team Earth from company D, as one of the new suppliers. The team practiced free seating and the seating arrangement for team Earth is shown in Figure 8.

4.3 Roles

It is necessary to outline how the different roles are carried out in this large-scale agile program since real-life cases offer from typical definition explanations. Table 9 presents an overview of the different roles in this program.

Table 9: An overview of the different roles in the program

Roles	Team Jupiter	Team Pluto	Team Mars	Team Saturn	Team Venus	Team Earth	Customer
PM 1	✓	✓	✓	✓	✓		
PM 2						✓	
PM 3							✓
Team Lead	✓	✓	✓	✓	✓		
Developer	✓	✓	✓	✓	✓		
UX Designer						✓	✓
PO Department		✓					
PO 1 Earth			✓				
PO 2 Earth					✓		
PO 3 Earth	✓			✓			
Technical Architect	✓	✓	✓	✓	✓	✓	
Solution Architect			✓				
Functional Architect						✓	✓
Test Lead							✓
Security Manager							✓

Project Manager (PM 1) DevOps Teams

The manager was the head and organizer for the DevOps teams and was responsible for coordinating meetings between the DevOps teams, team lead meetings, and present the company in different events. The manager role was also to coordinate demo meetings and be a part of the delivery of the products to the customer.

However, after the scaling process, the project manager resigned as manager. The former PM from team Earth took over this position and is now the head leader of the DevOps teams.

Project Manager (PM 2) Team Earth

Typical tasks a PM for team Earth work with is coordinating scheduled meetings and be a role between the supplier and the customer. The manager was hired in from a management consultancy company.

Project Manager (PM 3) Customer

The PM from the customer is the head leader for the project. This manager coordinates meetings, suppliers, stakeholders and participants in the project. The PM is responsible to lead meetings and has a high position in when it comes to decisions. The PM works closely with some of the hired architects and Product Owners.

Team Lead

The team lead is the head and organizer of the developers. The team lead manages and organizes internal meetings for the team, such as daily stand-up meeting and Sprint meeting. In the daily stand-up meeting, the team lead works as a Scrum Master. Other important tasks for the team lead is to ensure that the product is represented in a good way before putting it into production. Moreover, sometimes in some teams the team leads act also as a developer.

Developer

The developers in this large-scale agile program consist of both back-end and front-end developers. The developers work in a DevOps environment with responsibility in development and operations. Besides this, the developers also take care of the internal testing within the team.

UX Designer

The UX designer work with designing the solutions and often works closely with some of the architects. Some of the designers are hired designers from other technology companies, one is employed at Knowit, and one is from the customer. A representative from the customer leads the designers. Moreover, all the designers are located together as shown in Figure 8 next to team Earth.

Product Owner (PO)

The PO makes decisions regarding what are the most critical epics. The PO, in this case, is either from one of the hired consultants or one of the customers departments and they stand outside the DevOps teams. The hired PO's works also as an architect, one of them is a solution architect and the remaining PO's jobs as functional architects. The PO's decide not only crucial decisions. They also try to sell the products developed from the DevOps teams to several departments at the customer. The PO's can sell existing products or trade non-productive products. If the scenario is to sell non-productive products, the PO tells the team what to produce. As shown in Figure 15 the four different Product Owners, in this case, is mapped in an overview with relations to their teams.

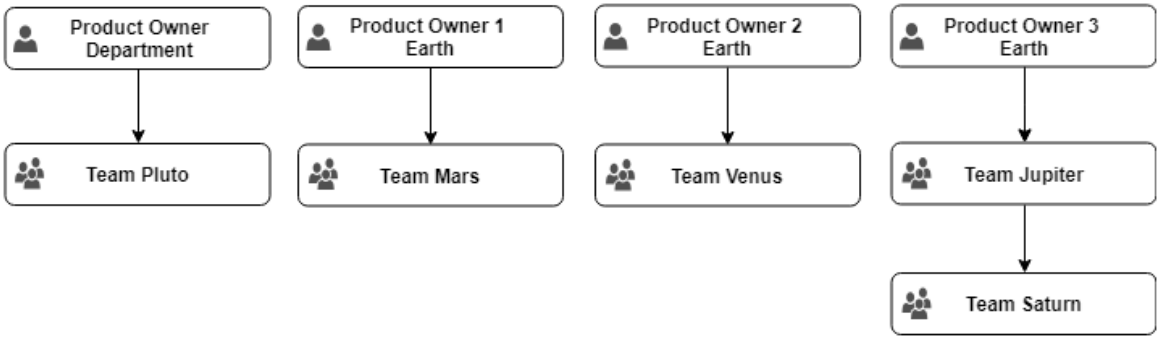

Figure 15: An overview of the Product Owner's relations to the DevOps teams

Architect

The architects, in this case, are employed at the customer, or the architects are representatives from the Product Owners. However, the difference between the type of architects is the knowledge and experience of the departments. Some architects work less with the departments and are more responsible for the product development process, including decision making and technical inputs. Some architects are located on-site, others are located off-site but attending important scheduled meetings on-site. The role architect in this study can be categorized into different roles: functional architect, solution architect and technical architect and supports the DevOps teams.

Test Lead

The test leader is responsible for thoroughly testing the solutions the teams create to find bugs. The test leader is employed at the customer and is located on-site with the members from the customer and team Earth.

Security Manager

The security manager is responsible for security testing of the solutions the teams create to find security risks. The security manager works closely with the test leader and is employed at the customer. The security manager is located off-site but attending important scheduled meetings on-site.

5 Results

In Chapter 4, the research context was described with an overview of the large-scale program. In this chapter, what was found through the data analysis will be presented, and the results is discussed in relation to the theory presented in Chapter 2.

Furthermore, this chapter describes different dependencies and their associated agile practices as coordination mechanisms using a taxonomy proposed by Strode (2016). The focus is to describe the various coordination mechanisms and dependencies because I examine what dependencies and their associated agile practices that facilitate the large-scale agile development.

In Chapter 2, I presented how Strode (2016) divided dependency into three main categories; *knowledge*, *process*, and *resource* (see Table 11). First, the *knowledge* dependency is split into expertise, requirement, task allocation and historical dependencies. Then, the *process* dependency into activity and business process dependencies. Lastly, the *resource* dependency into the entity and technical dependencies. Table 10 describes each of the dependencies. Table 11 shows an overview of the dependencies and the agile practices that act as coordination mechanisms in this large-scale program. 34 coordination mechanisms and 77 pairs of dependencies in total were found and the findings are based on my data described in Chapter 3.

Moreover, in Chapter 2, I presented how Strode et al. (2012) divided coordination mechanisms into eight main strategy components; *synchronization activity, synchronization artefact, boundary spanning activity, boundary spanning artefact, availability, proximity, substitutability* and *coordinator role*. Table 13 shows how I have mapped these strategy components with their mapped coordination mechanisms.

Table 10: A description of the dependencies (Strode, 2016)

Dependency		Description
Knowledge dependency	**Expertise**	Technical or task information is known only by a particular person or group and this affects, or has the potential to affect, project progress.
	Requirement	Domain knowledge or a requirement is not known and must be located or identified and this affects, or has the potential to affect, project progress.
	Task allocation	Who is doing what, and when, is not known and this affects, or has the potential to affect, project progress.
	Historical	Knowledge about past decisions is needed and this affects, or has the potential to affect, project progress.
Process dependency	**Activity**	An activity cannot proceed until another activity is complete and this affects, or has the potential to affect, project progress.
	Business process	An existing business process causes activities to be carried out in a certain order and this affects, or has the potential to affect, project progress.
Resource dependency	**Entity**	A resource (person, place, or thing) is not available and this affects, or has the potential to affect, project progress.
	Technical	A technical aspect of development affect progress, such as when one software component must interact with another software component and its presence or absence affects, or has the potential to affect, project progress.

5.1 Using the Taxonomy to assemble Agile Practices

Table 11: Dependencies and agile practices that act as coordination mechanisms in the large-scale program

Coordination mechanisms (34)	**Dependency**								
	Knowledge				**Process**		**Resource**		**Total**
	Expertise	**Requir-ement**	**Task allocation**	**Histo-rical**	**Activity**	**Business process**	**Entity**	**Tech-nical**	
Futurespective	✓								1
Scrum of Scrums	✓	✓	✓						3
Knowledge-sharing	✓	✓							2
Daily stand-up	✓		✓						2
One on one meeting		✓			✓	✓			3
Sprint		✓			✓	✓	✓		4
Software release								✓	1
Sprint Planning	✓	✓	✓						3
Task			✓						1
Informal ad hoc	✓	✓	✓		✓			✓	5
Team lead meetings				✓		✓	✓		3
Product backlog			✓		✓				2
Project meetings	✓		✓						2
Preparation for demo	✓			✓					2
Open work area		✓		✓					2
Wiki-Confluence		✓							1
Kanban board			✓		✓				2
Demo to customer		✓							1
Tools-Skype-Slack	✓		✓				✓		3
JIRA	✓		✓						2
Priority list	✓		✓						2
Wallboard		✓	✓		✓		✓		4
List of tests		✓							1
Whiteboard		✓							1
Full-time team	✓								1
Customer on-site		✓							1
Project manager	✓								1
Team lead/ SM	✓	✓	✓				✓		4
Test lead	✓	✓			✓			✓	4
Security lead	✓							✓	2
Product Owner	✓	✓	✓			✓			4
Technical architect	✓							✓	2
UX/designer	✓	✓							2
Scaling process	✓	✓		✓					3
Total Pairs of Dependency	**20**	**18**	**14**	**4**	**7**	**4**	**5**	**5**	
	56				**11**		**10**		**77**

To analyze which agile practices that act as coordination mechanisms to address dependencies and which dependency that was mostly used in the large-scale program, the data from Table 11 were merged. The agile practices are multipurpose because they can address more than a single dependency. Furthermore, Table 11 shows that 77 pairs of dependency in total was found with 34 different coordination mechanisms that act as agile practices.

The Frequency of Dependencies in the whole Large-Scale Program

A finding shown in Figure 16, is that 56 of 77 pairs of dependencies, or 73 % is knowledge dependency, 14 % is process dependency, and 13 % is resource dependency.

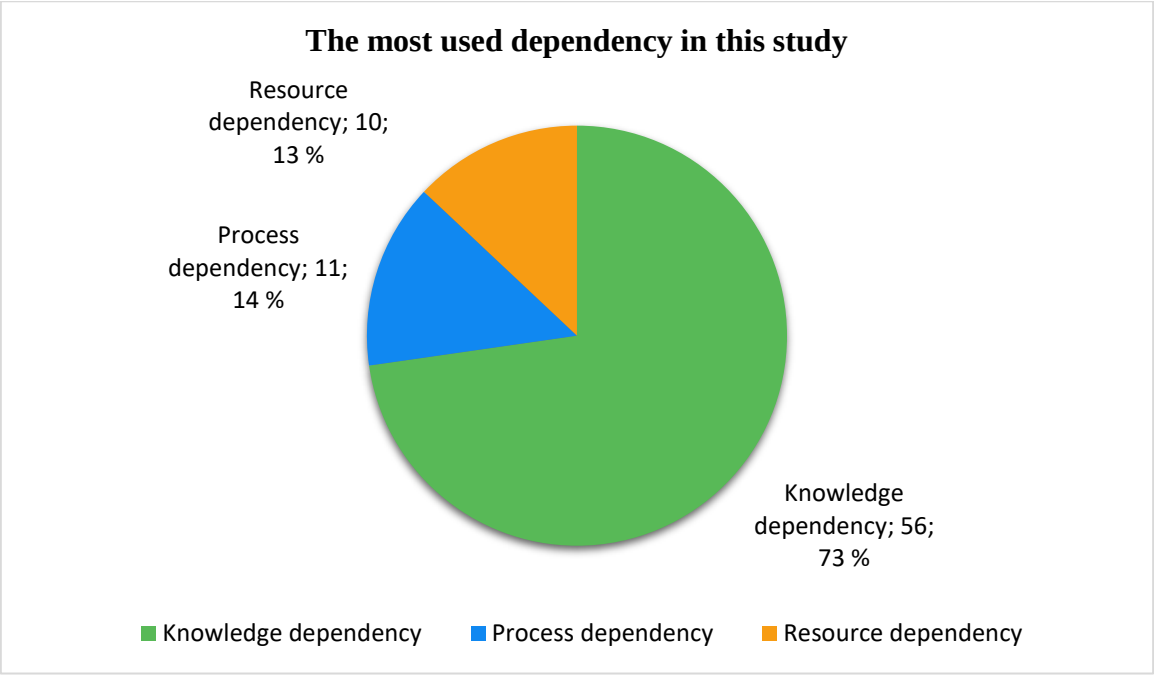

Figure 16: An overview of the frequency of dependencies in the whole large-scale program

Usage of Agile Practices

Since a finding around the most used dependency (knowledge dependency) in this study is presented, it could be interesting to look at the usage of agile practices that addressed the whole large-scale program and the most used dependency, knowledge dependency. A finding, shown in and Table 11, is that 20 different agile practices acted as coordination mechanisms to address the expertise dependency. This means that 20 of 34 coordination mechanisms, or 59 % of all the coordinative agile practices found in the whole large-scale program managed an expertise dependency.

Moreover, since knowledge dependency include 56 pairs of dependency (Table 11), and is clearly the largest used dependency, a more divided finding in knowledge dependency is presented. Shown in Figure 17 is that 20/56, or 36 % of all agile practices addressed in knowledge dependency is expertise dependency.

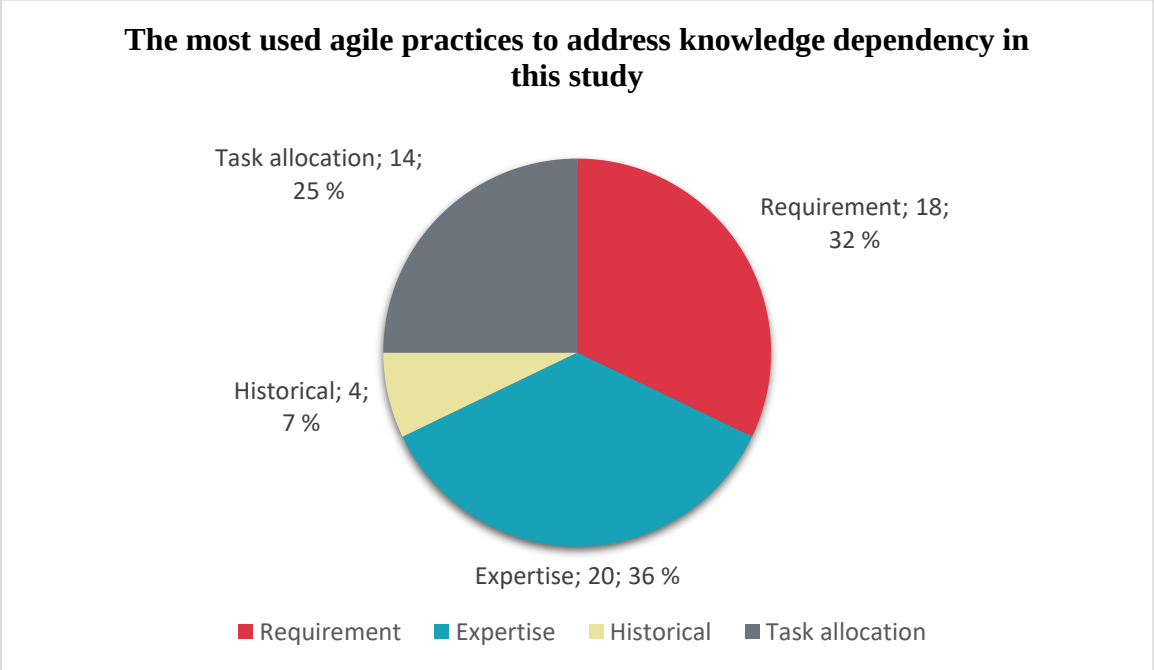

Figure 17:An overview of the agile practices usage in knowledge dependency

Used Agile Practices to Promote a Smooth Workflow in Large-Scale

To find the agile practices that promote a smooth workflow during the project in this large-scale program, Table 12 gives an overview of the agile practices that address three or more dependencies and therefore promotes a smooth workflow.

Table 12: The 12 agile practices found to address three or more dependencies

Agile practices	**Total of dependencies**	**Best matched dependency**
Informal ad hoc conversations	5	Knowledge dependency
Wallboard	4	Resource dependency
Team lead	4	Resource dependency
Test lead	4	Resource dependency
Product Owner	4	Process dependency
Sprint	4	Process dependency
Scrum of Scrum meetings	3	Knowledge dependency
One on one meetings	3	Process dependency
Sprint Planning meetings	3	Knowledge dependency
Team lead meetings	3	Knowledge dependency
Communication tools	3	Knowledge dependency
Scaling process	3	Knowledge dependency

Moreover, by this 12 recommended used agile practices to promote a smooth workflow in large-scale agile development, is it possible to match these coordination mechanisms to the best-fitted dependencies (Table 14), and look if the most used dependency actually affects. Figure 16 gives us an overview of the frequency of dependencies of total 77 pairs of dependencies mapped by 34 agile practices. The finding shows that the most used dependency was knowledge dependency with 73 % usage. If we look at the 12 agile practices showed in Table 12, we can see knowledge dependency is mapped six times, process dependency three times and resource dependency three times. This gives us this overview shown in Figure 18:

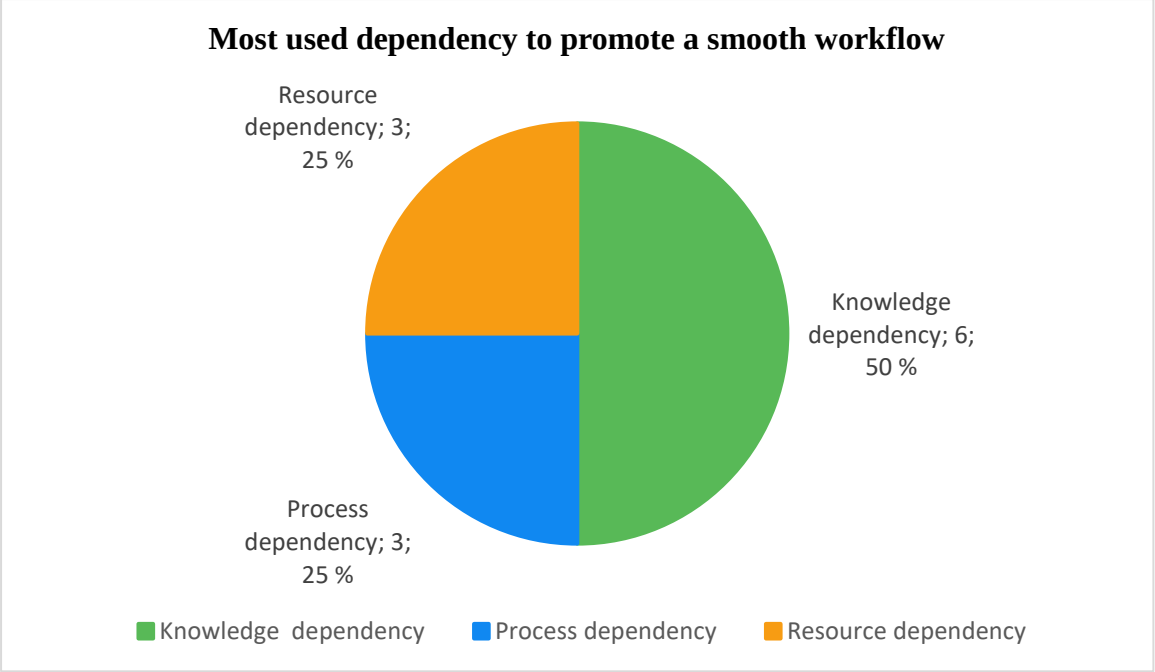

Figure 18: An overview of the frequency of dependencies in the 12 recommended agile practices

By this overview (Figure 18), we can see that the frequency number of agile practices map knowledge dependency for the highest usage in both the whole large-scale program with 73 % and in the recommended 12 agile practices to promote a smooth workflow in large-scale agile development with 50 %. With these findings, we can see that the most used dependency affects to promote a smooth workflow as well.

5.2 Dependencies and Coordination Mechanisms

Table 13: Dependencies and coordination mechanisms identified in the large-scale program

Knowledge dependency
Process dependency
Resource dependency

			Dependency							
						Knowledge		Process		Resource
	Strategy components		Expertise	Requirement	Task allocation	Historical	Activity	Business process	Entity	Technical
Coordination Mechanisms	Synchronization activities	Daily stand-up	■		■					
		Futurespective	■							
		Scrum of Scrum meetings	■	■	■					
		Software release								■
		Knowledge-sharing workshop	■	■						
		One on one meeting		■			■	■		
		Sprint Planning meetings	■	■	■					
		Sprint		■			■	■	■	
		Informal ad hoc conversation	■	■	■		■			■
		Team lead meetings				■		■	■	
		Project meetings	■		■					
		Preparation for product demo	■			■				
	Synchronization artefacts	Wiki		■						
		Kanban board			■		■			
		Task			■					
		Product backlog			■		■			
		Communication tool	■		■				■	
		JIRA	■		■					
		Priority list	■		■					
		Wallboard		■	■		■		■	
		Whiteboard		■						
	Boundary spanning activity	Product demo to customer		■						
		Informal ad hoc conversation		■						
	Boundary spanning artefact	List of tests		■						
		Scaling process	■	■		■				
	Availability	Full-time team	■							
	Proximity	Open work area		■		■				
		Customer on-site		■						
	Substitutability	Knowledge-sharing workshop	■	■						
	Coordinator role	Project manager	■							
		Team lead/ SM	■	■	■				■	
		Test lead	■	■			■			■
		Security lead	■							■
		Product Owner	■	■	■			■		
		Technical architect	■							■
		UX/ Designer	■	■						

Table 13 provides the data I mapped for this large-scale program and shows both dependencies and coordination mechanisms to illustrate how coordination mechanisms address dependencies. The mechanisms are also mapped to strategy components for following the taxonomy. The identified 34 agile practices are listed to dependencies, and strategy components and can address more than a single dependency. These dependencies, see Table 14, is described in section 5.3, 5.4 and 5.5 with some of the findings presented as agile practices that act as coordination mechanisms. The selected mechanisms are chosen because of the substantial, strong data collection on them, and minimum one mechanism for every dependencies will be described. Moreover, since some coordination mechanisms address more dependencies, the presented coordination mechanisms are selected for their best-matched dependency as shown in Table 14.

Table 14: The selected coordination mechanisms that will be described

Dependency		**Coordination Mechanisms**
5.3 Knowledge dependency	Expertise	Futurespective meeting Knowledge-sharing workshop Project meetings Project manager
	Requirement	Informal ad hoc conversations Product demo to customer Customer located on-site
	Task allocation	Daily stand-up meetings Scrum of Scrum meetings Sprint Planning meetings Kanban board Communication tools
	Historical	Team Lead meetings Open work area Scaling process
5.4 Process dependency	Activity	Sprint Wallboard
	Business process	Product Owner
5.5 Resource dependency	Entity	Team lead/ Scrum Master
	Technical	Test lead Security lead

5.3 Knowledge Dependency

I found that 33 of the 34 coordination mechanisms could be categorized as knowledge dependency, as they were meetings. Table 15 gives an overview of the coordination mechanisms that work as meetings or workshop in knowledge dependency, and Table 16 gives an overview of the roles of these meetings.

Table 15: The meetings identified in knowledge dependency

Coordination mechanisms	**Frequency**
Futurespective meetings	Rare
Knowledge-sharing workshop	Once a month
Project meetings	Every week
Product demo to customer	Every other week
Daily stand-up meetings	Daily
Scrum of Scrum meetings	Every week
Sprint Planning meetings	Every other week
Team lead meetings	Every week

Table 16: An overview of the roles of the meetings discovered as knowledge dependency

Attendees	**Future-spective**	**Work-shop**	**Project meetings**	**Demo**	**Stand-up**	**Scrum of Scrums**	**Sprint Planning**	**Team lead meetings**
Project manager			✓	✓		✓		✓
Team lead	✓	✓		✓	✓	✓	✓	✓
Developer	✓	✓		✓	✓		✓	
UX Designer			✓	✓		✓		
Product Owner	✓		✓	✓		✓	✓	
Architect			✓	✓		✓		
Test lead			✓	✓		✓		
Security manager			✓			✓		

5.3.1 Expertise Dependency

Typically, coordination mechanisms of expertise dependency include futurespective meetings, knowledge-sharing workshop, project meetings, and the role project manager. Expertise dependencies occurred in this large-scale program by these mentioned mechanisms. For example, a developer from team Venus noted that it was during the knowledge-sharing workshop he identified other knowledge and expertise about a security issue from other team members from other teams. This information made it possible to complete the security tasks during the Sprint.

Futurespective Meeting

The futurespective meeting has been recently introduced in the project. Team Pluto had a futurespective meeting in April 2018 with a department from the customer. This meeting was in cooperation between team Pluto and the department because the Product Owner from the department and the team lead from team Pluto wanted to gather all the representatives who had a role in the project. With team Pluto and the departments high level of cooperation and some internal issues which had occurred about planning, the management wanted to measure what could solve the problem. A team member stated:

> "*It is dangerous to ask for more predictable planning. Then it becomes a wish to follow the waterfall model. It is important to see that life has room for changes along the way.*"

This statements from the architect show that an approach to the agile development methodologies must be followed. Moreover, frustration about future deliveries was spread within the team, and a team member stated:

> "*We had to move many plans, it was frustrating because we had to look at what everyone else was doing.*"

Another agreed:

> "*We have too many activities at the same time, this can affect the workflow. There is little interaction with other teams relative to dates of deliveries.*"

To measure the problems, one solution was held by one futurespective meeting. One possible exercise in the futurespective meeting is called the hot-air balloon, where the team

members supposed success factors in the future. The team members wrote notes on post-it stickies with their thoughts about factors that will affect parts of the project's future, with either negative or positive notes. The positive notes were placed at the top of the balloon; this marked that the balloon got more air and could soar longer. The negative notes were placed at the bottom of the balloon; this was notes that could slow down the workflow in the project. A picture of this exercise is shown in Figure 19.

Figure 19: The hot-air balloon exercise in a futurespective meeting

Knowledge-Sharing Workshop

The developers from the DevOps teams meet once a month. The developers can attend voluntary upon request from the project manager. The developers who participate, get one hour to discuss and demonstrate front-end and back-end related programming code. The main goal in this type of workshop was to share skills among the team members. A typical scenario was when a front-end developer from team Jupiter demonstrated functions in the code to the rest of the developers. This was typically skills which may be as relevant to the other team members.

Project Meetings

The project meeting was a meeting typically for team Earth and the customers team members with a duration of 70 minutes once a week. The participants of this type of meeting were the project manager customer, project manager supplier, Product Owners,

test lead, architects, security manager and UX Designers. This meeting included status reviews on delivery from each competency areas. Typical topics that were raised during these types of the meeting was problems or other features that had occurred since the last meeting and the use of resources before new suppliers joined the project. It was the project manager from the supplier who delegated the words to the different participants. A project manager stated:

> "*The designers have to clarify its participation in the product team, and provide and introduce new designers, simultaneous prepare onboarding of new resources to the team.*"

To make sure that resources were complete before the scaling process, every part, such as DevOps teams and the designer team had to assemble their teams by decisions from team leads and managers with expertise experience.

Project Manager

I observed three project managers. One from the customer, one from team Earth and one from the supplier. The project manager from the customer had a main role in this project. His potential to lead meetings and discuss requirements were huge. By his expertise competence, the project manager from the customer had a position to make the final decisions about technical issues and tasks. A project manager stated during a demo meeting:

> *"For me personally, I am happy with the solutions. However, for the customer, we need to create a better user interface, so we follow the requirements set by the customer."*

The project manager roles are more described in Chapter 4 in section 4.9 Roles.

5.3.2 Requirement Dependency

Typically, coordination mechanisms in requirement dependency include informal ad hoc conversations, product demo to the customer and that the customer was located on-site. In the project, when developers or team leads from the DevOps teams failed on domain requirements, it was easy to talk to a specialist form the customer who was co-located in the room with the teams. This lead to easy communication when tasks on requirements

were failed or unclear. These situations often lead to ad hoc conversations. Based on evidence of this type, a requirement dependency is defined as a situation wherein domain knowledge is not known and must be located or identified, for example, when ad hoc conversations occurred between developers and project manager.

Informal Ad Hoc Conversations

Informal ad hoc conversations occurred several times a day. Every team in the project practiced this coordination mechanism. The informal conversations took place everywhere, especially where the teams were located. The open work area made it easy for the team members to make quick discussions, which created a fast working culture. By having other teams inside a short distance, it was possible to walk from team to team. The mechanism often occurred inside team Earth because they were located together with team members from the customer. One of the Product Owners stated:

> "*By sitting in the same location as the customer and a short distance from the DevOps teams it is possible to make important decisions through fast, informal conversations. If there are several small issues, it is easier to handle the problem by talking with other team members instead of using time on the issues in the scheduled meetings*."

Furthermore, the informal ad hoc conversations between other teams were observed most on the 4th floor because this area was the most open area and it was easy to take a short walk just around the corner.

Product Demo to Customer

The product demos were meeting related to the Sprint reviews. The meeting took place every other week at the end of a two-week Sprint with a duration of two hours. The purpose was to demonstrate the product to the customer. The demo meeting was divided into four parts; 30 minutes to team Venus, 30 minutes to team Jupiter, 30 minutes to team Mars and 30 minutes to team Pluto. Developers and team leads demonstrated the developed functionality to the project manager customer, Product Owners, teat lead, and designers. Discussion between designers and Product Owners often occurred in this type of meeting, and a designer stated:

"*We always have to follow standards from the requirements of the customer, functionality from one of the teams does not follow these requirements and need to be changed.*"

Moreover, one of the Product Owners stated:

"*These tasks are placed in the product backlog, and we always want to follow the customer requirements.*"

After the scaling process, and after team Saturn was introduced to the project, the demo meeting was changed. The four phases were merged, and the demos were not divided into DevOps teams anymore. Almost everyone from the project attended the meeting. This resulted in that 30 people attended the demo meeting after the scaling process.

Customer Located On-site

In the project, members from the customer was located on-site. This was roles as project manager, test lead, designer, and architects. The designer was located with the rest of the designers, and they created their own UX Designer team. By having the customer inside the head office, a project manager from the program stated:

"*By having the customer on-site and located together with us make the delivery fast, and it is possible to set the thing in production very quickly. The test responsible and the project manager can let us know in an easy way when deployment is allowed.*"

5.3.3 Task Allocation Dependency

Typically, coordination mechanisms in task allocation dependency include daily stand-up meetings, Scrum of Scrum meetings, Sprint Planning meetings, Kanban board and communication tools. In the project, tasks under development are displayed on a Kanban board. This is tasks grouped in a "to do" section, "in progress", "awaiting" and a "done" section. In the meetings, typically in the daily stand-up, Scrum of Scrums, and Sprint Planning, project team members always gave its status on tasks to other team members. This means that all the team members can see who is doing what and when. These task allocation dependencies are more described in detail in the following sections.

Daily Stand-Up Meetings

Team Venus started their stand-up every morning at 10:30 a.m., in which they followed up with lunch afterward. Team Venus carried out the meeting in their open workspace, next to their seats, and used a screen to involve team members on Skype who was not available at the office. This screen was the same screen used for wallboard. After the scaling process, team Venus got a new team lead. This team started their stand-up every morning at 09:30 a.m., and not followed up with lunch afterward. The team lead gave his reasoning and stated:

> "*The reason for starting the stand-up earlier is to sync the team members on where we stand with tasks. For me, it is a good routine to have a fixed time for these meetings, and the meetings help the team to kick-start the workday.*"

Meanwhile, team Jupiter started their stand-up every morning at 10:45 a.m., in which they followed up with lunch afterward. Team Jupiter carried out the meeting in their open workspace by standing up from their seats. During the meeting, all the computer screens at the desks were standing in between the team members. Furthermore, team Mars did it in the same way as team Jupiter, but started their stand-up every morning at 11:15 a.m., and followed up with lunch afterward. Team Saturn started their stand-up every morning at 10:30 a.m., in which they followed up with lunch afterward. Since team Saturn took over team Venus earlier workspace, Saturn carried out the meeting in their open workspace, next to their seats with a screen available.

Sequence of Categories

For daily stand-up in all observed teams, it was a typically common sequence of interaction in the daily meetings. I analyzed the interaction processes at the meetings and identified the various patterns of a sequence of interaction. The sequence was also the pattern that included all three Scrum questions (Q1, Q2, Q3) (C1). The most common sequence of interaction at the meetings is shown in Figure 20. A description of the coding schema is placed in Appendix B (Stray et al., 2012).

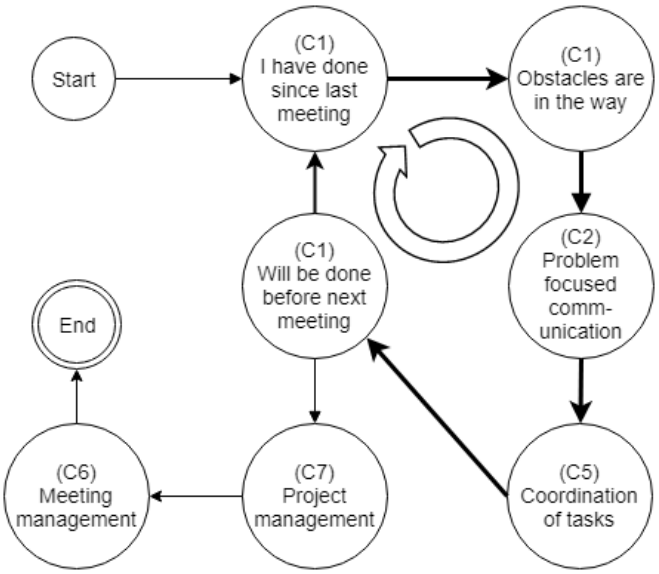

Figure 20: The most common sequence of interaction in a daily meeting

When following this pattern, a team member started by telling the team what he or she had done since last daily stand-up meeting (C1) before discussing obstacles (C1). An obstacle often caused another team member to discuss problem focused communication (C2) regarding answer on what is the best solution to the identified problem. The discussion usually ended up with coordinating tasks (C5) with a discussion of who should be involved in solving the task and the obstacles. To the end, the team member ended his or her round by telling what will be done before next meeting (C1). Then the next team member started his or her status update, and the cycle began again by telling what was done since last meeting (C1). After the update from the last team member, they provided information about other meetings (C7) before summing up the meeting (C6) and ended it.

Scrum of Scrum Meetings

Scrum of Scrums meeting was a meeting with a duration of 60 minutes and was held every week. The participants in this meeting were project manager supplier and customer, team leads, UX Designers, Product Owners, architects, test lead, and security manager. Moreover, maximum 12 participants attended the meetings. Furthermore, every team lead gave a status of their tasks to the project manager customer. The team leads were prepared and got an own list of topics on what they are going to say in the meeting. During the meeting, the participants discussed issues on what the team leads submitted. The Product Owners and the project managers discussed the issues with the specific team lead and tried

to make a good solution on what the DevOps team could do to solve the problem. The team lead always noted the suggestions from the Product Owner and the project manager.

A project manager from the program stated following about the Scum of Scrum meetings:

> "*We schedule this type of meeting every week because it is possible to gather different important roles from the project. It also allows us to prepare and discuss problems when roles, such as Product Owners and team leads are gathered together. This makes it worth spending one hour weekly on this type of meeting.*"

The Scrum of Scrum meetings observed can be categorized as fully distributed Scrum (Lee & Yong, 2009). With findings on that the teams were cross-functional with members from several different locations, such as security manager and architects located off-site, and Product Owners, UX Designers, project managers and test lead gathered together as one large unit for coordinating tasks.

Sprint Planning Meeting with Venus

At the Sprint Planning meeting team Venus divided the meeting into two phases; pre-planning and Sprint Planning. First, 60 minutes with pre-planning was conducted. The purpose of the pre-planning was to plan unfinished tasks from the last Sprint and plan new tasks for the new Sprint period. The team lead focused on goals and tasks from the "to do" list from the Kanban-board and discussed the tasks with the team members. Together, the team members coordinated the tasks to each other and estimated time for when the task should be done. If the task was not given priority, the task was placed in the product backlog. This ensured that Sprint Planning meeting ensured coordination between the team. Then, after the pre-planning, the team followed up with lunch.

Lastly, after the lunch, the team met for Sprint Planning. The duration was new 60 minutes with the developers, the team lead, and the Product Owner. The goal was to submit the tasks with the Product Owner that was discussed in the previous pre-planning. If the Product Owner disagreed with the estimates of the tasks, the Product Owner overturned the tasks and changed the priority list and the product backlog.

Kanban Board – Jira[2]

Jira creates user stories, issues, plan sprints, prioritize, and distribute tasks across teams. The team lead, and the Product Owner was responsible for updating the Kanban board in Jira and entered tasks into Jira. The team estimated the time for each task and an example of the Kanban board is shown in Figure 21. The Kanban board was an important coordination mechanism during Sprints and meetings, such as Sprint Planning meetings and project meetings.

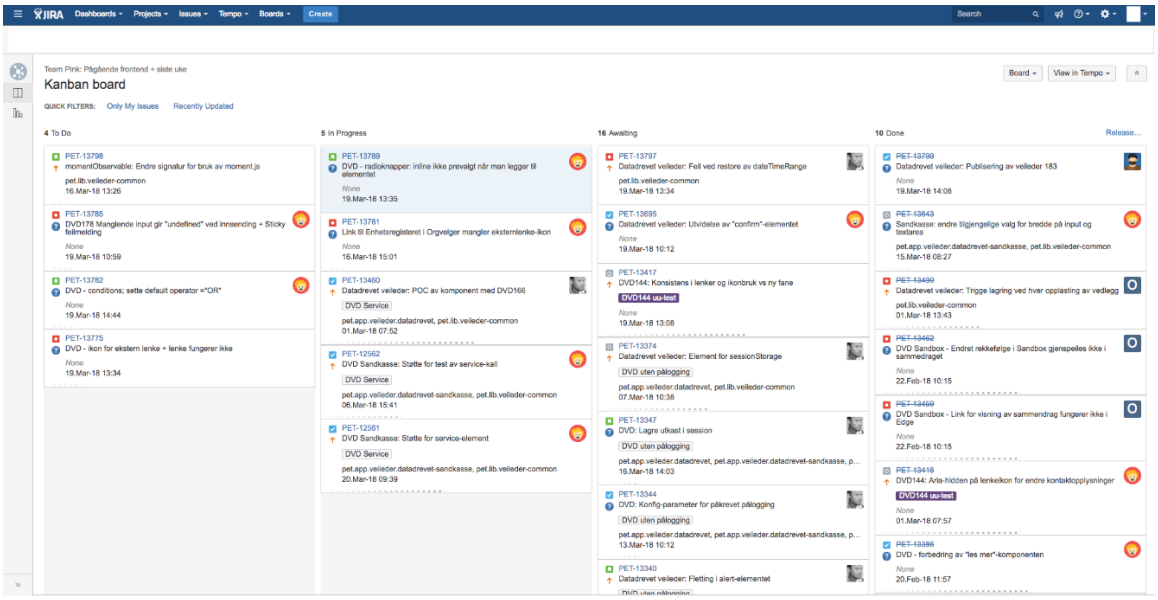

Figure 21: The Kanban board in Jira

The tasks in the Kanban board was grouped in a "to do" section, "in progress", "awaiting" and a "done" section. When the teams were going to updating and adding tasks to the Kanban board, the team leads, and the Product Owners were the responsible for the implementing.

Communication Tools – Slack[3] – Skype[4]

The teams made use of several tools for communications, such as Slack and Skype. Slack was used for communications through channels. The channels divided the tasks and grouped them in a clearer overview when messaging. The communication tool informed the team members about deliveries and other work-related tasks. In addition, the tool was

[2] Jira is a registered trademark of Atlassian, www.atlassian.com/software/jira
[3] Slack is a registered trademark of Atlassian, www.slack.com
[4] Skype is a registered trademark of Skype Technologies and parent is Microsoft, www.skype.com

also rather social, by inviting other teams to lunch and other social small events during the workday.

Skype is a tool for video conversations and chats messaging. This tool was used when team members joined meetings, such as daily stand-up and demo meetings through video if they were located at home or in a special trip away from the office.

5.3.4 Historical Dependency

Typically, coordination mechanisms in historical dependency include team lead meetings, open work area, and scaling process. Historical dependencies were identified in this large-scale program by decision making around mechanisms. The decisions around the mechanisms were typical of persons with many years experience and historical knowledge about a mechanism. The scaling process was determined by people in the management, persons with the historical background of such large projects. Based on evidence of this type, a historical dependency is defined as a situation wherein knowledge about past decisions is needed. For example, team leads discussed about resources in their team in the team lead meetings.

Team Lead Meetings

The team lead meeting was a meeting with a duration of 60 minutes once a week. The participants during this type of meeting were all team leads, and the project manager form the supplier. The topics for the meeting was internal situations and to give status updates for each team. Every team lead talked about what was done since last team lead meeting and updated the project manager about finished tasks. It was also normal to discuss about resources in each team, and this was a challenging topic before the scaling process that introduced three new suppliers. The team leads knew little about what would happen with their teams. A team lead from one of the teams with a high historical competence stated:

> "*I don’t know what happens to my team. If the plan is to continue, we need more resources in the form of back-end developers. We don’t want that one of our team members shall be removed because our team has too many priorities in the project. We also do not want resources from the new suppliers if we should be able to finish our tasks but introduce resources from our own company with historical competence.*"

Moreover, another team lead from the program stated:

> "*We also need more and stronger resources if we shall reach the goals and complete our tasks in this project.*"

The project manager also gives updates on what was done at the management meetings the day before. Issues and information about resources from the management in the company was shared with the team leads.

Open Work Area

The DevOps teams, team Earth and the customer team was seated in an open work area. Figure 22 provides a picture of the work area. The open work area facilitated coordination through easy access to other team members and teams and enabled quick oral coordination. The work area where an area for discussion topics about tasks and solutions and allowed the coordination mechanism informal ad hoc conversation. Moreover, with meeting rooms available just a few meters from the seating arrangements, it was possible to implement informal and unscheduled meetings.

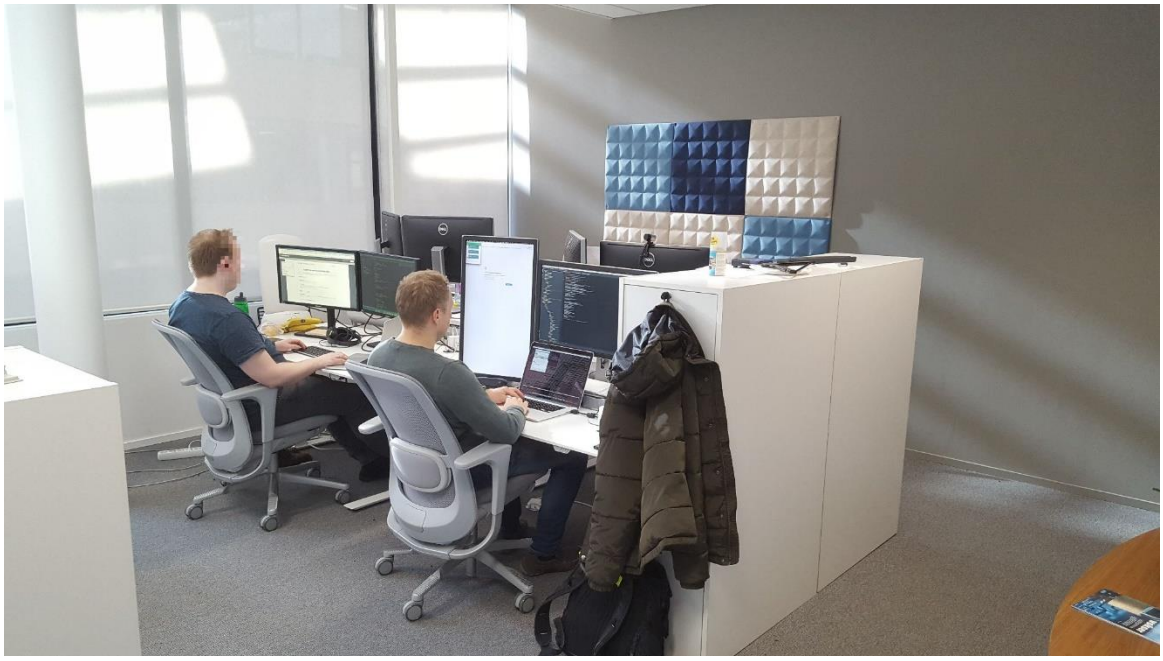

Figure 22: The open work area from a DevOps team

In large projects, it is many participants involved and is therefore impossible to gather all members from the project, and we know that the location influences the coordination. The project manager from the supplier stated following about this mechanism:

"*By using the offices on the best possible way, and so many participants involved in the project, we know from earlier that open areas make the teams more autonomous and enable decisions in the project.*"

Scaling Process

From the 1st of January 2018, the customer introduced three new suppliers to the project. This process reorganized the large-scale program with new team members, such as developers, team leads, architects, and designers. The DevOps teams went from being four teams to five teams. The onboarding period lasted in around two months, and the project management introduced a trial period with some reorganizing in the meetings and the teams, and lastly, introduced a new type of meetings. The trial period was introduced to the new coordination mechanisms took place in an easy way for the employees. A member of the project management stated:

"*We did not decide who should be the supplier, the customer regulates this. By letting three new suppliers join the project is perfect for the process and the workflow. By doing this the teams are more cross-functional, and we can solve some of the problems we had with the business model.*"

5.4 Process Dependency

I found that 9 of the 34 coordination mechanisms could be categorized as process dependency.

5.4.1 Activity Dependency

Typically, coordination mechanisms of activity dependency include the scrum practice Sprint and the tool wallboard. In the project, activity dependencies occurred. During the Sprints, activities in the form of tasks could not proceed until another activity was completed. The wallboard coordinates this activity dependency by sharing the Kanban board at the team locations.

Sprint

A Sprint was conducted over a two-week period. This was two weeks with the development of tasks and product increment. At the start of a Sprint, Sprint Planning meetings were conducted with the team members, team lead, and the Product Owner. At the end of a Sprint, product demo meetings were conducted by product demonstration to the customer. This meeting was the last ceremony in a Sprint period for the five DevOps teams.

Wallboard

To make sure tasks enabled for the teams, wallboards were placed on the 4th and 5th floors. The wallboard was big TV screens and was used to display activities and project progress by a dashboard.

5.4.2 Business Process Dependency

Typically, coordination mechanisms of business process dependency include the role product owner. Business process dependency occurred in this project by roles that improved a solution to the public by selling to different departments.

Product Owner

In this study, it was observed four Product Owners. Some of them also acted as architects and had a relation to the DevOps teams by attending meetings such as Sprint Planning meetings, demo meetings and Scrum of Scrum meetings. The Product Owner also tried to sell finished solutions to different departments at the customer. The role Product Owner is described more in detail in Chapter 4 in section 4.9 Roles.

5.5 Resource Dependency

I found that 10 of the 34 coordination mechanisms could be categorized as resource dependency.

5.5.1 Entity Dependency

Typically, coordination mechanisms in entity dependency include the role team lead. Entity dependency occurred in this program by physical things such as people.

Team Lead/ Scrum Master

During the observations, it was observed four team leads. The team leads also acted as Scrum Masters during the daily stand-up meetings. For letting the project be in process, developers were an object that was required. When the DevOps teams missed this requirement, it was the team lead that ensured that all resources were obtained. The team lead role is more described in Chapter 4 in section 4.9 Roles.

5.5.2 Technical Dependency

Typically, coordination mechanisms in technical dependency include the roles test lead and security manager. Technical dependencies occurred in the project when the security manager during the project meetings told the project members that critical issues had to be addressed to the Security Council and the test lead missed test data from the developers.

Test Lead

During the observations, it was observed one test lead. The test lead ensured that one software component interacted with another software component and the test lead was a required object for this project. The role test lead is more described in Chapter 4 in section 4.9 Roles.

Security Manager

During the observations, it was observed one security manager. The security manager ensured that software components corresponded to the security requirements. The role security manager is more described in Chapter 4 in section 4.9 Roles.

6 Discussion

In chapter 5 the results based on my data described in Chapter 3 was outlined. In this chapter, the discussions from the results are outlined, such as discussions of the theory and related research to answer the research questions in this study.

The results in Chapter 5 show the agile practices that acted as coordination mechanisms in a large-scale agile program with seven different teams, six of them located on-site and one of them located off-site. The results showed 34 mechanisms and 77 mapped pairs of dependencies. The dependencies were described by their best matched agile practices with a description of the practices. Later, by using the dependency taxonomy by Strode (2016), it was possible to assemble the usage of agile practices, find the frequency of dependencies and promote a smooth workflow in large-scale agile development by suggesting 12 agile principles.

Before answering the research questions, I will discuss whether the participants in the program belongs to a working group or a team. This part is necessary because it will influence the next discussion sections in this chapter. To discuss this topic, I will use the theory by Katzenbach and Smith (2005) described in Table 5 in section 2.4.1 in Chapter 2.

Furthermore, to answer the research questions, I will first discuss and analyze my results on dependencies and coordination mechanisms compared with other studies and discuss what dependencies that occur in large-scale agile. Second, discuss and explain my suggested agile practices in large-scale which can form a smooth workflow and could be a recommended starter set for providing coordination compared with suggested agile practices in co-located. Lastly, the implications for practice and theory will be discussed.

6.1 Working Group or a Working Team

The participants in the program belong to a group or a team. To answer this topic, I use the proposed theory by Katzenbach and Smith (2005). Table 17 shows the comparison between a working group and a working team from the analyzed program.

Table 17: The observed program, a working group vs. a team

Working group	**Yes**	**Team**	**Yes**
Strong, clearly focused leader		Shared leadership roles	✓
Individual accountability		Individual and mutual accountability	✓
The group's purpose is the same as the broader organizational mission	✓	Specific team purpose that the team itself delivers	✓
Individual work products		Collective work products	✓
Runs efficient meetings	✓	Encourages open-ended discussion and active problem-solving meetings	✓
Measures its effectiveness indirectly by its influence on others (such as financial performance of the business)		Measures performance directly by assessing collective work products	
Discusses, decides, and delegates		Discusses, decides, and does real work together	✓
Sum	2		6

By analyzing and summarizing Table 17, the participants of the analyzed program match both the characteristics of a working group and a team. Moreover, the "team" part got six matches, and seem to be dominant. The program matches the characteristics of a working team. First, the teams had shared leadership roles represented by the project managers and the team leads. Second, the teams had individual and mutual accountability, and the specific teams purposed that the teams itself delivered. Third, the teams in the program encouraged open-ended discussions and conducted problem-solving meetings through project meetings, Scrum of Scrum meetings and daily stand-up meetings. Lastly, the teams discussed, decided and did real work together by following each team's area of responsibility.

However, the participants in the program belong to working teams instead of working groups. This is helpful when discussing the next sections in this chapter, and it realizes that the large-scale agile program consists of seven teams and not groups. Moreover, the "collective work products" section in Table 17 is marked since the teams worked on the

same solution. The information that one team had about their solution was relevant for the others, for example through demo meetings and Scrum of Scrum meetings.

6.2 Dependencies and their associated Agile Practices that facilitate the Large-Scale Agile

This section answering and discuss the first research question:

RQ1: *What dependencies and their associated agile practices that act as coordination mechanisms facilitate the large-scale agile development?*

In Chapter 2 different case studies and theory were presented. This section will look at the findings in the case studies presented in Chapter 2 in relation to my case study about coordination mechanisms in the large-scale agile development program. The goal is to present my results and findings compared with other findings in other case studies, with a discussion about findings that acts as agile practices in large-scale development related to knowledge dependency, process dependency and resource dependency as shown in Table 18.

Table 18: Agile practices from this study compared with other findings in other studies

Dependency	**Agile practices**
Knowledge dependency	Scrum of Scrum meetings
	Daily Stand-up meetings
	Sprint Planning meetings
	Product demo to customer
Process dependency	Sprint
	Product Owner
Resource dependency	Team lead/ Scrum Master
	Test lead

6.2.1 Knowledge Dependency

The aggregated category of knowledge dependencies accounted for 73 % of all dependencies across the project and was the most used dependency. By mapping agile practices to the best-matched dependency, it was possible to categorize the agile practices to one main dependency. Agile practices in knowledge dependency that could be interesting to compare with other findings in other case studies are the following: *Scrum of Scrums meetings*, *daily stand-up meetings*, *Sprint Planning meeting*, *product demo to customer* and *scaling process*.

Scrum of Scrum Meetings

The goal of Scrum of Scrum meetings is to allow teams to communicate with each other to ensure that the solutions integrate well with the fundamentals of the other teams. The meeting is suggested to be time-boxed to last a maximum of 15 minutes (Larman & Vodde, 2010), just like the daily stand-ups. Other indicates to last 30-60 minutes (Cohn, 2007). Challenges in Scrum of Scrums is not to make it into a status reporting meeting for management, but to keep it as a synchronization meeting between the teams (Larman & Vodde, 2010).

Paasivaara et al. (2012) found that Scrum of Scrums in two large-scale projects worked with at least 20 teams each was challenging. The representatives were from all teams with roles including managers, Scrum Masters, architects, Product Owners, developers, and testers. The results showed that the audience was too big to keep everyone interested, and the participants did not know what to report. Moreover, as a result to this, one of the case projects introduced feature-specific Scrum of Scrums meetings for 3-5 teams. This new introduction seemed to work well, but challenges with coordination at the project level remained.

However, the use of Scrum of Scrums seemed to work well in my case study. The weekly meeting gathered the whole project participants, expect the developers. This compressed the meeting considerably, but team members form seven teams maintained still the number of participants. The study by Paasivaara et al. (2012) worked well after the introduction of involving just 3-5 teams. This compressed the meeting, but every participant from the teams still attended. In my case study, seven teams participated as mentioned, but without the developers, this reduced the number of participants. Maximum 12 participants participated

in the meetings. That we can see from this, it can be essential to compress the participants so that the meeting can involve everyone. In my case study, this worked well, and a distributed Scrum occurred. This meeting was an excellent arena to allow teams to communicate with each other and integrate information and knowledge from other teams, simultaneously give status about tasks to the project manager. This makes the Scrum of Scrums meeting to fit the task allocation dependency (knowledge dependency). Everyone was interested, and 60 minutes was enough to involve everyone.

Furthermore, I would argue and suggest allowing maximum 60 minutes of duration of a Scrum of Scrums meeting. 15 minutes, like the daily stand-ups is a short time to involve everyone. 60 minutes and few numbers of participants allows discussion if significant problems showing up and situations like in the study by Paasivaara et al. (2012), where the participants do not know what to report, disappears.

Daily Stand-up Meetings

The goal of the daily stand-up meeting is to involve every team members and let them speak to three questions (Q1, Q2, Q3 from Chapter 2, section 2.1.1). Stray et al. (2016) found that daily stand-up meeting affects more than we think by analyzing data from four countries, 12 software teams, 60 persons and 79 observed daily stand-up meetings. The study shows that the meeting should be held on time before lunch and it very important to be standing during these meetings because to be standing shorter down the time.

Similarly, the daily stand-up meeting in my case study seemed to be implemented in the same way from the literature from Stray et al. (2016) case study. In my case study, four of five teams followed up with lunch. They also practiced the standing method by speak to the three suggested questions (Q1, Q2, Q3), but in a different order. In my case, Figure 20, in Chapter 5, showed the interaction process for the daily stand-up meetings and identified the various patterns of a sequence of interaction. I found that the frequencies for the questions (Q1, Q2, Q3), were followed in the order; Q1, Q3 and then Q2. Moreover, I expected the sequence to follow the Scrum guide, which turned out and was not the case.

In knowledge dependency, I would argue that the use of daily stand-up meeting should be implemented as a recommended coordinated mechanism by letting team members share status with the rest of the team members. The daily stand-up meeting allows the team

members share information on who is doing what, and when. This makes the mechanism to fit the task allocation dependency (knowledge dependency).

Sprint Planning Meetings

The goal in Sprint Planning meeting is to delegate tasks to team members, estimate time on the tasks, and make a priority list in the Sprint. Abrahamsson et al. (2017) suggested that the meetings should be divided into two phases: First, users, management, the customer, and the Scrum team held the meeting to decide goals for the next sprint. Second, Scrum Master and the Scrum team focusing on how the product increment is implemented during the Sprint. In my case study it was completed in two phases, but in the opposite order that Abrahamsson et al. (2017) suggested. First, pre-planning with the team lead/ Scrum Master and the developers were gathered. Second, the Product Owner, team lead, and developers were gathered. The team lead submitted the tasks with the Product Owner to the developers.

Moreover, to facilitate a proper coordination mechanism in knowledge dependency, I suggest following the two-phased layout from my case study. The team needs to involve the customer (PO) once. By including the customer once, it lets the Product Owner in my case study release time for other tasks, and he can be involved in other Sprint Planning meetings because he maybe is related to other teams that also ends and starts the Sprint period at the same time. I would argue that the Sprint Planning meeting fit the task allocation dependency (knowledge dependency) because the meeting gives an overview who is doing what and when.

Product Demo to Customer

Demonstrating the functionality of the software is done by a product demo meeting. The teams are located together with the customer and demonstrate the functionality. Nyrud and Stray (2017) found that a demo meeting facilitated coordination because it was an arena for creating common expectations and the meeting created a common understanding of the developed product. Moreover, in my case study, the demo meeting also created a good coordination mechanism by creating expectations and created a common understanding of the product, between the teams and the customer. This common understanding of the project fit the requirement dependency (knowledge dependency) by giving a common

domain knowledge of the project. Therefore, my findings from the observations, are compared to Nyrud and Stray (2017).

Paasivaara et al. (2009) found the biggest problem with the demo meeting was the use of technology, teleconferencing and application sharing. This did not offer enough possibilities to communicate efficiently. However, from the observations, this seemed to be unproblematic in my case. The customer was located on-site, and a scheduled practice was used to conduct demo meetings at the end of every Sprint. The project manager from the customer always communicated well by giving feedback to each team after the demonstration. By this, an efficient communication culture was solved.

Moreover, Rolland et al. (2016) found that the demo meeting was improvised in the middle of Sprints to get the users feedback. In my case study, this was done oppositely. The demo meeting was performed after each Sprint, every second week. My findings, therefore, did not compare in the same way as the guidelines for tailoring agile in the large-scale.

6.2.2 Process Dependency

In process dependency, which covered 14 % of agile practices, it could be interesting to compare following agile practices to other case studies: *Sprint* and *Product Owner*.

Sprint

In the project, a Sprint period was scheduled for two weeks. The two-week Sprint was weeks with the development of tasks and product increment. Cooper and Sommer (2018) also looked at how the Sprint period was conducted in agile development projects form six different case studies. They found at the beginning of each Sprint the development team met to agree on what it can accomplish in the Sprint and created a task plan by a Sprint Planning meeting. During the Sprint, daily stand-up was held to ensure that work is on course to accomplish in the last 24 hours, and what should be done in the next 24. At the end of each sprint, product demo meetings and retrospective meetings were held to review how team members worked together. Moreover, the practices during the Sprint period as described in the article by Cooper and Sommer (2018) increased in my case study as well, expect the mechanism retrospective.

I would argue that the Sprint mechanism is an important recommended artifact in Scrum since my findings and resent study on the mechanism Sprint seems to work well in agile development programs. Moreover, I argue that the mechanism maps the activity dependency (process dependency) well, because a Sprint is the main period for completing tasks (activities) before other tasks can proceed.

Product Owner

In the project, the role Product Owner attended Sprint Planning meetings, project meetings, Scrum of Scrum meetings and demo meetings. The Product Owner was related to one or more DevOps teams, but was not a team member in practice, the Product Owner was standing outside and presented team Earth. In the study by Bass (2015) the Product Owner identified and prioritized customer requirement, and the Product Owner was formed into a team. In the case study, Bass identified nine team functions for the Product Owner, in my case six functions were found. My findings on the Product Owners functions compared with Bass (2015) is shown in Table 19.

Table 19: The functions of the role Product Owner in my study and Bass (2015)

This study	Bass (2015)
Prioritize tasks	Groom
Technical architect	Prioritize tasks
Functional architect	Release master
Communicator	Technical architect
Translating business needs	Governor
Business seller	Communicator
	Traveler
	Intermediary
	Risk assessor

Both results show that the Product Owner also has the role as an architect. In my case as a technical architect or a functional architect and prioritizing tasks. I would argue that the role Product Owner is a defined role with given functions matched to the project type and size and fit the process dependency. By mapping the Product Owner, the Product Owner fit the description of the business process dependency (process dependency), by causing activities in form of tasks in the project.

6.2.3 Resource Dependency

The aggregated category of resource dependencies accounts for 13 % of all dependencies across the project. Agile practices in resource dependency that could be interesting to compare with other findings in other case studies are the following: The roles *team lead,* and *test lead*.

Team Lead/ Scrum Master

In my case study, the team lead also worked as a Scrum Master and was the head and organizer of the developers. The team lead managed and organized internal meetings for the team, such as daily stand-up meetings and Sprint Planning meetings. Other important tasks for the team lead was to ensure that the product is represented in a good way before putting it into production. Compared with findings from other studies by Bass (2014), T. Dingsøyr et al. (2018) and my study, the role Scrum Master comprises activities as shown in table 20.

Table 20: Functions for a Scrum Master in my study, Bass (2014) and T. Dingsøyr et al. (2018)

This study	Bass (2014)	T. Dingsøyr et al. (2018)
Stand-up facilitator	Process anchor	Stand-up facilitator
Sprint planner	Stand-up facilitator	Iteration planning
Release facilitator	Impediment remover	Demonstration facilitator
Developer	Sprint planner	Retrospective facilitator
Resource responsible	A Scrum of Scrums facilitator	
	Integration anchor	

I would argue that the role Scrum Master fit the entity dependency (resource dependency) because the Scrum Master is a resource in the form of a person. In my case, the Scrum Master ensures that the teams were prepared with resources in the way of developers. This resource, the developer, was an object that was required for the project progress.

Test Lead

In my case study, the test lead was responsible for thoroughly testing the solutions the teams create to find bugs and ensuring that one software component interacted with another software component. In the case study by Torgeir Dingsøyr et al. (2018), they found that

the test lead made sure that testing was conducted at team level by unit tests, integration tests, system tests, and system integration tests.

Moreover, I would argue that the role test lead fit the technical dependency (resource dependency) because the test lead is an object that is required for the project progress and makes sure that one software component interacts with another software component by integration tests and system integration tests.

6.2.4 What Dependencies occur in Large-Scale Agile Program

My case study provided evidence for eight types of dependencies, including expertise, requirements, historical, task allocation, activity, business process, entity and technical dependencies. For example, does the knowledge dependency involve knowing how to do an activity or task (expertise), by the mechanism project manager. Knowing what to do (requirement), for the mechanism product demo. Knowing who is doing what and when (task allocation), by the mechanism Kanban board. Alternatively, knowing how or why things were done in the past (historical), by the mechanism team lead meetings.

Moreover, by having identified dependencies, it is useful to understand which dependencies that occurred most frequently in the program. By doing this, the most common dependency should have an impact on large-scale agile program coordination. In my case study, the most frequently occurred dependency was knowledge dependency, with 73 % coverage. This coverage is an exciting finding because it indicates that addressing knowledge dependencies should have an impact on large-scale agile program coordination.

By only addressing knowledge dependency and setting focus on the dependency with the highest frequency, is an interesting statement. But what about the critical findings from the process and resource dependencies, such as the mechanisms Sprint, Product Owner, Scrum Master (team lead in my case) and test lead? These mentioned mechanisms are mapping the process and resource dependencies. These two dependencies were mapped together with 27 % overall coverage of agile practices. The role Scrum Master, for example, is an essential mechanism in large-scale development projects and is mapped to the resource dependency. By look at the comparing with my study, Bass (2014) and T. Dingsøyr et al. (2018) about the functions associated with the Scrum Master, the results shows that the Scrum Master has different but essential tasks in during projects. By this, there is necessary

not to forget mechanisms connected to other dependencies, such as process dependency and resource dependency than just knowledge dependency in large-scale agile programs.

6.3 Providing Coordination in the Large-Scale Agile

6.3.1 Implications for Practice

This section answering and discuss the second research question:

RQ2: *What could be a recommended starter set for providing coordination in large-scale agile development programs by using a dependency taxonomy?*

I will discuss my study's 12 suggested agile practices that map a starter set for providing coordination in large-scale agile development with findings from agile practices in co-located projects by Strode (2016). The 12 suggested agile practices have identified that address three or more dependencies (see Table 11), and these practices are potentially a useful minimal set for coordinating a large-scale program. Table 21 gives an overview of my suggested agile practices and suggested findings from co-located that could be a recommended starter set for providing coordination:

Table 21: Agile practices from my study and co-located by Strode (2016)

Large-scale agile practices (this study)	**Agile practices in both (this study and Strode (2016))**	**Co-located agile practices (Strode (2016))**
Team lead Test lead Product Owner Scrum of Scrum meetings One on one meetings Sprint Planning meetings Team lead meetings Communications tools Scaling process	Informal ad hoc conversations Sprints Wallboard	Cross-team talk A co-located team Iteration planning session Story breakdown session A done checklist Working software at the end of each sprint A single priority team A product backlog User stories for managing requirements

The dependencies in agile software development can help teams, team members and other participants involved in development projects to choose valid coordinative practices

(Strode, 2016). Since agile practices map dependencies, it is essential to look at the agile practices in use and implement these practices as coordination mechanisms. In large-scale, I would argue that my suggested 12 agile practices are an excellent recommended starter set for providing coordination. But there is one essential and central mechanism I think missing both in large-scale and in co-located, the daily stand-up meeting. Stray et al. (2017) invited professional developers of a programming forum for a survey and obtained 221 responses. They found that 87 % of those who practice agile methods in their projects used daily stand-up meetings. They argued that the value of the meeting should be evaluated according to the team needs.

My Suggested Agile Practices Divided in Scrum Types

It would be interesting to discuss my findings on the suggested agile practices that providing coordination in a recommended starter set in the large-scale agile. Table 22 sows my agile practices mapped to a Scrum type.

Table 22: My agile practices mapped to a Scrum type

Type	**Practices**
Roles	Team lead Test lead Product Owner
Ceremonies	Scrum of Scrum meetings One on one meetings Sprint Planning meetings Team lead meetings
Tools	Communication tools
Incident	Scaling process

Roles

When implementing the roles in the software industry, I recommend applying a team lead, a test lead and a Product Owner in a large-scale project. In my case study, these roles had a strong position in the project and were significantly involved at team-level. These roles lead to a high value of coordination. Moreover, what about a project manager? I would argue that the role project manager should be implemented such as a team lead and a Product Owner. By using the dependency taxonomy by Strode (2016), the project manager

is not a recommended implemented role. However, still, in my case study, the project managers lead to coordination because of their knowledge, historical background, and the network was used.

Ceremonies

The four ceremonies in my case study worked well, and I will argue that the use of the dependency taxonomy by Strode (2016) to find the suggested ceremonies. I recommend implementing the meetings: Scrum of Scrum meeting, Sprint Planning meeting, and team lead meeting. The mentioned meetings were meetings that effectively created coordination in the large-scale agile program. Furthermore, I still recommend implementing the daily stand-up meetings, although the dependency taxonomy mapped the mechanism two times (see Table 11). The daily stand-up in this program created coordination and discussed tasks and internal issues at team-level.

Tools

For implementing tools in the daily work, I recommend the use of Slack. This tool lets team members communicate digitally, and it is possible to join channels. The channels let the team members communicate effectively by messaging through proper channels divided into roles, topics or teams.

Incident

Another interesting finding in my study is the mechanism scaling process. By following the dependency taxonomy by Strode (2016), the scaling process mechanism was mapped three times (see Table 11). By this mapping, it makes the mechanism to be one of the 12 mechanisms that providing well coordination in large-scale programs. Furthermore, in my case, the scaling process lead to weakness. The earlier project manager for the DevOps teams was sent off the program, and the project manager from team Earth took over her position. This process reduced the number of project managers, and the program lost valuable knowledge, and her network disappeared. For the software industry, the scaling process mechanism should be read with some degree of skepticism.

6.4 Implications for Theory

Taxonomy by Strode (2016)

The dependency taxonomy by Strode (2016) was implemented to identify dependencies and coordination mechanisms in the large-scale program. Further research in the case study by Strode (2016) was to assess the applicability of the taxonomy in a context such as large-scale agile. The taxonomy was used as a framework to identify coordination, and I want to say that it seems to be successful. The taxonomy let the coordination mechanisms be mapped in their best-matched strategy components and dependencies. By this, it is possible to collect out the best used agile practices in a project and suggest the practices in similar projects in the software industry.

Limitations

The results presented in the compared case studies are collected in several work units with interviews and observations over a longer period, while my results are based on only one case project based on a taxonomy used as a framework to map my findings during observations. Therefore, the results should be read with some degree of skepticism. There are other frameworks that could have been chosen to map the findings, such as the theory proposed by Van De Ven et al. (1976).

Validation

Since the use of a theory increases the external validity, the taxonomy of dependencies and coordination mechanisms proposed by Strode (2016) was used to getting an overview of the field of dependencies and coordination mechanisms. On the data collection, I followed the four strategies of data collection and fieldwork purposed by Patton (2002) since the construct validity is concerned about the relation between the general investigated phenomenon and the specific data. Furthermore, the conversations that I had with team leads and team members, where they corrected me during the end of the observation period, reduced the threat to construct validity. The conversations contributed a better understanding of the program and created a match between the general phenomenon that was investigated and the measurement. Moreover, the internal validity is not relevant as my study is not trying to examine causal connections.

7 Conclusion and Future Work

In this master thesis, the goal was to investigate what dependencies and their related agile practices that acted as coordination mechanisms to facilitate the large-scale agile development. Then, what could be a recommended starter set for providing coordination in the large-scale agile development program by using a dependency taxonomy. In order to answer the research questions, I observed 40 meetings to get insight into the company ways of working in a large-scale software project. The program involved seven autonomous teams, five of the these were DevOps teams.

The case study explored agile practices that acted as coordination mechanisms. To map the different coordination mechanisms, a dependency taxonomy was used. The dependency taxonomy was useful for describing different dependencies and their associated agile practices to achieve effective project coordination to tailoring the large-scale agile program. The coordination mechanisms made collaboration between the teams in the program, by implement Scrum of Scrum meetings, daily stand-up meetings, demo meetings, Sprint Planning meetings, and introduce different roles, such as project managers, team leads, Product Owners and DevOps developers. These mechanisms lead to fast Sprint periods, frequent production setting, a common understanding of what is being created, and autonomous decisions in the program.

Moreover, the case study mapped coordination mechanisms into their best-matched coordination strategy components, and the mechanisms were multipurpose because they addressed more than a single dependency. In addition, the case study found that knowledge dependencies are predominant. Moreover, to benefit coordination in large-scale agile development programs, I suggest that focusing on selecting agile practices that address the types of knowledge dependency.

12 agile practices can address multiple project dependencies. This would be a good starter set of practices for programs to achieve effective coordination and support collaboration. The 12 agile practices are: Informal ad hoc conversations, wallboard, team lead, test lead, Product Owner, Sprint, Scrum of Scrum meetings, one on one meetings, Sprint Planning meetings, team lead meetings, communication tools and scaling process. Overall, these suggested agile practices would lead to providing coordination in the large-scale agile development and hopefully provide a smooth workflow in projects.

For future work, I recommend doing the same work as in this thesis with other datasets and comparing the outcome. Further research could, therefore, be to use the dependency taxonomy to discover more effective agile practices in large-scale agile development programs to create better coordination in the software industry. Second, it is recommended to look at the decision making in agile software development. Further research could, therefore, be to look at how is it possible to create coordination through decisions.

Bibliography

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile Software Development Methods: Review and Analysis. *VTT Technical Research Centre of Finland, VTT Publications 478*.

Ahmad, M. O., Markkula, J., & Oivo, M. (2013). *Kanban in software development: A systematic literature review.* Paper presented at the 2013 39th Euromicro Conference on Software Engineering and Advanced Applications.

Anderson, D., Concas, G., Lunesu, M. I., & Marchesi, M. (2011). Studying Lean-Kanban Approach Using Software Process Simulation. *Agile Processes in Software Engineering and Extreme Programming*, 12-26.

Bang, S. K., Chung, S., Choh, Y., & Dupuis, M. (2013). *A grounded theory analysis of modern web applications: knowledge, skills, and abilities for DevOps*. Paper presented at the Proceedings of the 2nd annual conference on Research in information technology.

Bass, J., Beecham, S., Nic Canna, C., Noll, J., & Razzak, M. A. (2018). *A Large Empirical Study of the Product Owner Role in Scrum*.

Bass, J. M. (2014). *Scrum Master Activities: Process Tailoring in Large Enterprise Projects.* Paper presented at the 2014 IEEE 9th International Conference on Global Software Engineering.

Bass, J. M. (2015). How product owner teams scale agile methods to large distributed enterprises. *Empirical Software Engineering, 20*(6), 1525-1557.

Beranek, G., Zuser, W., & Grechenig, T. (2005). Functional group roles in software engineering teams. *SIGSOFT Softw. Eng. Notes, 30*(4), 1-7.

Cao, L., & Ramesh, B. (2007). Agile Software Development: Ad Hoc Practices or Sound Principles? *IT Professional, 9*(2), 41-47.

Cohn, M. (2007). Advice on conducting the scrum-of-scrums meeting.

Cooper, R. G., & Sommer, A. F. (2018). Agile–Stage-Gate for Manufacturers. *Research-Technology Management, 61*(2), 17-26.

Coyle, S., Conboy, K., & Action, T. (2015). An Exploration of the relationship between Contribution Behaviours and the Decision Making Process in Agile Teams. *International Conference on Information Systems*.

Crowston, K., & Osborn, C. S. (1998). A Coordination Theory Approach to Process Description and Redesign. *Management Sciences and Quantitative Methods*, 1-60.

Denison, D. R., Hart, S. L., & Kahn, J. A. (1996). From Chimneys to Cross-Functional Teams: Developing and Validating a Diagnostic Model. *Academy of Management Journal, 39*(4), 1005-1023.

Dingsøyr, T., Fægri, T. E., & Itkonen, J. (2014). *What Is Large in Large-Scale? A Taxonomy of Scale for Agile Software Development.* Paper presented at the International Conference on Product-Focused Software Process Improvement, Cham.

Dingsøyr, T., Moe, N. B., Fægri, T. E., & Seim, E. A. (2018). Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. *Empirical Software Engineering, 23*(1), 490-520.

Dingsøyr, T., Moe, N. B., & Seim, E. A. (2018). Coordinating Knowledge Work in Multi-Team Programs: Findings from a Large-Scale Agile Development Program. *Project Management Journal*.

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology, 50*(9), 833-859.

Erickson, J., Lyytinen, K., & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: The state of research. *Journal of Database Management, 16*(4), 88-100.

Evbota, F., Knauss, E., & Sandberg, A. (2016). *Scaling up the Planning Game: Collaboration Challenges in Large-Scale Agile Product Development*, Cham.

Fink, L., & Neumann, S. (2007). Gaining Agility through IT Personnel Capabilities: The Mediating Role of IT Infrastructure Capabilities. *Journal of the Association for Information Systems, 8*(8), 440-462.

Fitzgerald, B., & Stol, K.-J. (2014). *Continuous software engineering and beyond: trends and challenges*. Paper presented at the Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, Hyderabad, India.

Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software, 123*, 176-189.

Gerring, J. (2007). *Case Study Research: Principles and Practices*. New York: Cambridge University Press.

Gregor, S. (2006). The Nature of Theory in Information Systems. *MIS Quarterly, 30*(3), 611-642.

Jain, R., & Suman, U. (2017). An Adaptive Agile Process Model for Global Software Development. *International Journal on Computer Science and Engineering, 9*.

Janz, B. D., Wetherbe, J. C., Davis, G. B., & Noe, R. A. (1997). Reengineering the Systems Development Process: The Link between Autonomous Teams and Business Process Outcomes. *Journal of Management Information Systems, 14*(1), 41-68.

Jarzabkowski, P. A., Lê, J. K., & Feldman, M. S. (2012). Toward a Theory of Coordinating: Creating Coordinating Mechanisms in Practice. *Organization Science, 23*(4), 907-927.

Johannessen, A., Tufte, P. A., & Christoffersen, L. (2010). *Introduksjon til Samfunnsvitenskapelig Metode* (4 ed.): Abstrakt forlag AS.

Katzenbach, J. R., & Smith, D. K. (2005). The discipline of teams. *Harvard Business Review, 83*(7), 167.

Kirk, D., & MacDonell, S. G. (2015). Progress report on a proposed theory for software development. *10th International Joint Conference on Software Technologies (ICSOFT)*, 1-7.

Kornilova, I. (2017). DevOps is a culture, not a role! Retrieved from https://medium.com/@neonrocket/devops-is-a-culture-not-a-role-be1bed149b0

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

Lee, S., & Yong, H.-S. (2009). Distributed agile: project management in a global environment. *Empirical Software Engineering, 15*(2), 204-217.

Lwakatare, L. E., Karvonen, T., Sauvola, T., Kuvaja, P., Olsson, H. H., Bosch, J., & Oivo, M. (2016). *Towards DevOps in the Embedded Systems Domain: Why is It So Hard?* Paper presented at the 2016 49th Hawaii International Conference on System Sciences (HICSS).

Malone, T. W. (1988). *What is Coordination Theory?* .

Malone, T. W., & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Comput. Surv., 26*(1), 87-119.

Marczak, S., & Damian, D. (2011). How interaction between roles shapes the communication structure in requirements-driven collaboration. *2011 IEEE 19th International Requirements Engineering Conference*, 47-56.

Merriam, S. B. (1998). *Qualitative research and case study applications in education*. San Francisco, CA: Jossey-Bass.

Miles, M. B., & Huberman, A. M. (1994). *Qualitative Data Analysis: An Expanded Sourcebook*: Thousand Oaks: Sage.

Moe, N. B., Dingsøyr, T., & Dybå, T. (2010). A teamwork model for understanding an agile team: A case study of a Scrum project. *Information and Software Technology, 52*(5), 480-491.

Moe, N. B., Holmström Olsson, H., & Dingsøyr, T. (2016). *Trends in Large-Scale Agile Development: A Summary of the 4th Workshop at XP2016*. Paper presented at the Proceedings of the Scientific Workshop Proceedings of XP2016, Edinburgh, Scotland, UK.

Munassar, N. M. A., & Govardhan, A. (2010). A Comparison Between Five Models Of Software Engineering. *International Journal of Computer Science Issues, 7*(5), 94-101.

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the Association for Computing Machinery, 45*(5), 72-78.

Nitto, E. D., Jamshidi, P., Guerriero, M., Spais, I., & Tamburri, D. A. (2016). *A software architecture framework for quality-aware DevOps*. Paper presented at the Proceedings of the 2nd International Workshop on Quality-Aware DevOps.

Nyrud, H., & Stray, V. (2017). *Inter-team coordination mechanisms in large-scale agile*. Paper presented at the Proceedings of the XP2017 Scientific Workshops, Cologne, Germany.

Ohno, T. (1988). *The Toyota production system; beyond large-scale production*: New York: Productivity Press.

Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2009). *Using Scrum in Distributed Agile Development: A Multiple Case Study.* Paper presented at the 2009 Fourth IEEE International Conference on Global Software Engineering.

Paasivaara, M., Lassenius, C., & Heikkilä, V. T. (2012). Inter-team coordination in large-scale globally distributed scrum: do scrum-of-scrums really work? *Proceedings of the 2012 ACM-IEEE international symposium on Empirical software engineering and measurement, 19*(20), 235-238.

Parker, G. M. (2003). *Cross- Functional Teams: Working with Allies, Enemies, and Other Strangers, Edition 2*. San Francisco: Jossey-Bass.

Patanakul, P., Chen, J., & Lynn, G. S. (2012). Autonomous Teams and New Product Development. *Journal of Product Development & Management Association, 29*(5), 734-750.

Patton, M. Q. (1990). *Qualitative Evaluation and Research Methods* (2 ed.). Beverly Hills, CA: SAGE Publications.

Patton, M. Q. (2002). *Qualitative Research & Evaluation Methods* (3 ed.). California: SAGE Publications.

Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). Capability maturity model, version 1.1. *IEEE Software, 10*(4), 18-27.

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., & Still, J. (2008). The impact of agile practices on communication in software development. *Empirical Software Engineering, 13*(3), 303-337.

Platt, J. (1992). "Case Study" in American Methodological Thought. *SAGE Social Science Collections, 40*(1).

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Crawfordsville, Indiana, US: Addison-Wesley.

Pries-Heje, L., & Pries-Heje, J. (2011). *Why Scrum Works: A Case Study from an Agile Distributed Project in Denmark and India.* Paper presented at the 2011 Agile Conference.

Raman, S. (1998). *Lean software development: is it feasible?* Paper presented at the 17th DASC. AIAA/IEEE/SAE. Digital Avionics Systems Conference. Proceedings (Cat. No.98CH36267).

Rolland, K. H., Mikkelsen, V., & Næss, A. (2016). *Tailoring Agile in the Large: Experience and Reflections from a Large-Scale Agile Software Development Project*, Cham.

Schmidt, C. (2016). *Agile Software Development Teams*. München, Germany: Springer International Publishing Switzerland.

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

Schwaber, K., & Sutherland, J. (2013). The Scrum Guide. Retrieved from http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf

Spradley, J. P. (1980). *Participant Observation* (1 ed.). Austin, TX: Holt, Rinehart and Winston.

Stray, V., Moe, N. B., & Aurum, A. (2012). *Investigating Daily Team Meetings in Agile Software Projects.* Paper presented at the 2012 38th Euromicro Conference on Software Engineering and Advanced Applications.

Stray, V., Moe, N. B., & Bergersen, G. R. (2017). *Are Daily Stand-up Meetings Valuable? A Survey of Developers in Software Teams.* Paper presented at the International Conference on Agile Software Development, Cham.

Stray, V., Sjøberg, D. I. K., & Dybå, T. (2016). The daily stand-up meeting: A grounded theory study. *Journal of Systems and Software, 114*, 101-124.

Strode, D. E. (2012). *A Theory of Coordination in Agile Software Development Projects.* (Doctor of Philosophy), Victoria University of Wellington,

Strode, D. E. (2016). A dependency taxonomy for agile software development projects. *Information Systems Frontiers, 18*(1), 23-46.

Strode, D. E., Huff, S. L., Hope, B., & Link, S. (2012). Coordination in co-located agile software development projects. *Journal of Systems and Software, 85*(6), 1222-1238.

Sutherland, J., & Schwaber, K. (2007). The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process. *Co-Creators of Scrum*.

Van De Ven, A. H., Delbecq, A. L., & Koenig, R. (1976). Determinants of Coordination Modes within Organizations. *American Sociological Review, 41*(2), 322-338.

Wiedemann, A. (2018). IT Governance Mechanisms for DevOps Teams – How Incumbent Companies Achieve Competitive Advantages. *Proceedings of the 51st Hawaii International Conference on System Sciences*, 4931-4940.

Wohlin, C., Šmite, D., & Moe, N. B. (2015). A general theory of software engineering: Balancing human, social and organizational capitals. *Journal of Systems and Software, 109*(Supplement C), 229-242.

Yin, R. K. (2002). *Case study research: Design and methods*. California: SAGE Publications

Appendix

Attachment A: Observation Protocol

Topic	Questions
Space	What is the layout of the physical room? How are the actors positioned?
Participants	What are the names and relevant details of the people involved? Is someone acting as a leader or facilitator?
Activities	What are the various activities and discussions?
Objects	Which physical elements are used?
Acts	Are there any specific individual actions? What are the ways in which all actors interact and behave toward each other?
Events	Are there any particular occasions or anything unexpected?
Time	When does the meeting start? What is the sequence of events? When does the meeting end?
Goals	What are the actors attempting to accomplish?
Feelings	What are the emotions in the particular contexts? How is the atmosphere?
Closing	How is the meeting ended? Is there a post meeting?

(Spradley, 1980; Stray et al., 2016)

Attachment B: Coding Scheme for Daily Stand-up

No	Category	Explanation	Examples
C1	The three Scrum questions	What have I done since the last meeting? What will be done before the next meeting? What obstacles are in the way?	Informing other members about the tasks the individual is working on.
C2	Problem focused communication	The major questions and problems that need to be addressed, including the elaboration of the issue and discussions of possible solutions to the problems.	Discussing questions such as "How can we implement that feature? How do we integrate the components? What is the best solution to the identified problem?"
C3	Clarification	Explanations that make an earlier event, situation, or statement clear.	Questions that someone asked to better understand an issue.
C4	Criterion	The reasons or arguments that evaluate an alternative solution or proposal.	Arguments for a solution, customer requirements, and technical possibilities.
C5	Coordination of tasks	Delegation of tasks and assignment of responsibilities.	Discussing who should be involved in solving a task.
C6	Meeting management	Statements related to the orchestration of the meeting activity.	Indicating that members hold off on discussions, asking someone to speak, and summing up the meeting.
C7	Project management	Statements concerning activities not directly related to the content of the meeting.	Discussing the reporting of resources and hours and providing information about other meetings.
C8	Digression	Discussions of side topics or interruptions related to things outside the content of the meeting.	Telling a joke, interrupting because of technical problems with phone conference equipment, and other distractions

(Stray et al., 2012)