Managing dependencies in agile DevOps

A case study of coordination

Emilie Mæhlum

Thesis submitted for the degree of
Master in Informatics: Programming and System Architecture
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2020

Managing dependencies in agile DevOps

A case study of coordination

Emilie Mæhlum

Managing dependencies in agile DevOps

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

Printed: Reprosentralen, University of Oslo

Abstract

Background: Coordination is a crucial aspect of agile software development. Managing dependencies is important for efficient coordination. Understanding what activities and artifacts manages different dependencies can help companies coordinate better and choose the mechanisms best suited for their coordination needs.

Aim: This thesis aims to investigate how an agile DevOps is managing dependencies to achieve effective coordination. This is examined by identifying dependencies and coordination mechanisms for managing these dependencies. Additionally, the aim is also to investigate which barriers for managing dependencies can be found in an agile DevOps context.

Method: A qualitative case study was conducted. Data were collected by conducting 9 interviews, observing 32 workdays, and observing 49 meetings. Additionally, various documents and chat logs were collected.

Results: The results revealed 38 coordination mechanisms and 95 pairs of dependencies present in the development team. These coordination mechanisms manage knowledge dependencies, process dependencies, and resource dependencies. Also, working remotely, role clarity, planning, and estimation, and implementing changes in the software development process was identified as barriers to managing dependencies.

Conclusion: It is possible to use a dependency taxonomy to identify coordination mechanisms and dependencies in an agile DevOps company. The most crucial coordination mechanisms found in the company included the Zendesk planning meeting, the daily stand-up, ad hoc conversations, the sprint planning meeting, and communication tools because they managed four or more dependencies.

Acknowledgements

Writing this master thesis has been a challenging, yet rewarding experience. Completing this thesis would not have been possible if it wasn't for a number of people. Firstly, I am incredibly thankful to my supervisor Viktoria Stray for her invaluable guidance, extensive knowledge and tremendous support and enthusiasm. Furthermore, I am extremely grateful to all the participants of this study, for welcoming me into their company and including me in their activities and workplace as one of their own. Without the openness, this study would not have been possible. I would like to thank all my fellow students and the employees of the research group Programming and Software Engineering for valuable discussions, distractions and help.

I would like to thank family and friends for unparalleled support and encouragement throughout the process. A special thanks to my great friend Kristine for making every day at the university fun. I would also like to thank my boyfriend Ivar for all the patience, love and support. Lastly, I am especially and immensely thankful to my grandmother for the endless encouragement she has given me.

Emilie Mæhlum
July, 2020

Contents

List of Figures

List of Tables

1 Introduction

Coordination is a crucial, yet challenging area of agile software development. Malone and Crowston, 1994 defines coordination as "the managing of dependencies between activities". Managing dependencies within companies is important for efficient coordination. Dependencies can be managed well, poorly or not at all, but when managed well, it suggests that the right coordination mechanisms are present (Strode, 2016).

Agile development methods and DevOps require adaptation during implementation to meet the needs of a constantly changing software development environment (Hemon, Fitzgerald, Lyonnet, & Rowe, 2020), which makes the management dependencies a continuous effort. Understanding how to manage dependencies in agile projects may help product leaders, managers, and developers to create better agile behaviors and more successful projects by choosing appropriate coordination practices from the large number of agile practices that exist (Stray, Moe, & Aasheim, 2019). However, few studies have focused on managing dependencies in an agile DevOps context.

This thesis seeks to bridge the gap in research by studying dependency management and coordination mechanisms in an agile DevOps company, as well as examining barriers for efficient coordination.

1.1 Motivation

The main motivation for this thesis came from the fact that I was interested and fascinated by the company I have chosen to study. I thought their product was impressive and refreshing, but I was also interested in their use of technology and how they work. I was curious about how they coordinate, especially because the company was located in multiple locations.

1.2 Research Area and Questions

The research area of this thesis is coordination in an agile DevOps company. Malone and Crowston, 1994 define coordination as "the managing of dependencies between activities." The process of coordination will be studied by examining the coordination mechanisms within the company.

There are a variety of coordination mechanisms, such as meetings, roles, and tools. Many of the coordination mechanisms are related to the software development methodology, such as daily stand-up, sprint planning meetings, and backlog grooming, this leading to parts of the agile software development practices being examined.

The research questions are:
RQ1: *How are dependencies managed in an agile DevOps company?*
RQ2: *What are barriers for managing dependencies in an agile DevOps company?*

1.3 Approach

A case study was conducted to answer the research questions through the use of relevant theories. The data used in this study consists mainly of observation and interviews, and the data were analyzed in a qualitative data analysis tool called NVivo. This analysis tool helped structure the data collected into manageable data sets, which was analyzed analyzed and used to answer the research questions.

1.4 Chapter Overview

Chapter 2: Background gives a brief introduction to *agile software development methodologies, coordination, teams, technical debt and startups.*

Chapter 3: Research Method outlines the research method, research design, data collection and validity of the study.

Chapter 4: Research Context presents an overview over the setting where the data collection took place.

Chapter 5: Results presents the results found in relation to the theory presented in Chapter 2.2.

Chapter 6: Discussion contains discussion of the results in relation to the research questions and findings from other research.

Chapter 7: Conclusion and future work presents the conclusion to the research questions and propose what future work could focus on.

2 | Background

This chapter will present the background and theory relevant to this thesis. Here I will give an introduction to the relevant software development methodologies, as the methodology affects the coordination by providing ceremonies and various areas for coordination. Second, theories on coordination are presented. Then, a theoretical background about teams is presented together with a theory on team classification. Lastly, an introduction to Technical debt, Start-ups, and DevOps are presented as they are relevant for the research context.

2.1 Software Development Methodologies

2.1.1 Agile Software Development

Agile software development represents a new approach for planning and managing projects by focusing on flexibility, complexity, change, and uniqueness, rather than up-front plans and strict plan-based control (Nerur & Balijepally, 2007). Agile values and principles, as presented in the Agile Manifesto, are:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation

- **Responding to change** over following a plan

2.1.2 Lean Software Development

Lean, or Lean thinking, is adapted from Toyota's Lean manufacturing. Its main focus is on creating value for the customer, eliminating waste, optimizing value streams, empowering people, and continuously improving (Ebert, Abrahamsson, & Oza, 2012). Lean 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. 1). Poppendieck and Poppendieck, 2003 presents the seven principles of Lean as the following:

1. Eliminating waste
2. Amplifying learning
3. Decide as late as possible
4. Deliver as fast as possible
5. Empower the team
6. Build integrity in
7. See the whole

2.1.3 Scrum

Scrum is an agile and iterative approach for developing products and services (Larman & Vodde, 2008; Rubin, 2012). It structures development in time-boxes called Sprints, where each Sprint is 1 to 4 weeks (Larman & Vodde, 2008). Sprints have a fixed duration with a specific start date and an end date. A Sprint works as a development cycle. It is not a standardized process, but a framework for organizing and managing work where the key structural components should be the Scrum values,

principles, and practices (Rubin, 2012). One important principle is to "inspect and adapt". This is done by taking a short step of development and examine both the resulting product and the current practices, and based on this adapting the product goals and processes (Larman & Vodde, 2008).

Scrum practices involves the following roles, activities and artifacts (Rubin, 2012, p. 14):

- **Roles:** Development Team, Scrum Master and Product Owner
- **Activities:** Sprint, Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective and Product Backlog grooming
- **Artifacts:** Product Backlog, Sprint Backlog, Increment

The Scrum Team The Scrum team is made up by the three roles in Scrum: the development team, the scrum master, and the product owner. This team typically consists of 5-9 people. Scrum Teams are self-organizing and cross-functional, meaning that they are autonomous and have all the skills and knowledge needed to complete the tasks (Larman & Vodde, 2008; Schwaber & Sutherland, 2017).

Scrum Master The Scrum Master is responsible for facilitating Scrum in the development project. In other words, the responsibility of the Scrum Master is to help everyone understand and follow the process of Scrum. The Scrum Master is not a project leader and does not have any authority over the team, but is a servant leader of the team and protects them from outside interference. The Scrum Master can not be the same person as the Product Owner, but can be a person from the Development Team (Larman & Vodde, 2008).

Product Owner The Product Owner is responsible for maximizing the value of the product resulting from the work of the Development Team (Schwaber & Sutherland, 2017, p.6). The Product Owner acts as a link between the Scrum Team and

the stakeholders. The Product Owner must understand the needs and priorities of the stakeholders, the customers, and the users to represent them in the development and ensure that the right product is developed (Rubin, 2012). This can include identifying product features, translating these into a list of tasks, choosing the tasks for the next Sprint, and continually re-prioritizing and refining the list (Larman & Vodde, 2008). In Scrum, only one person is serving as the Product Owner (Larman & Vodde, 2008; Rubin, 2012).

Development Team The Development Team is responsible for developing and delivering an increment of "done" product at the end of a sprint (Schwaber & Sutherland, 2017, p.7). The Development Team is the only one who can deliver such an increment of "Done" product. The Team is cross-functional and has all the expertise needed to deliver an increment at the end of each sprint (Larman & Vodde, 2008). Development Teams are also self-organizing and manage their work in the way they see fit.

Sprint In Scrum, work is performed in time-boxed iterations called Sprints. Sprints are of the same length throughout the development process and include Scrum activities such as Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective, and the development (Schwaber & Sutherland, 2017). Each sprint has a specific goal of what is to be built and delivered as an increment. When a Sprint ends, a new Sprint starts immediately.

Sprint Planning Each Sprint starts with a Sprint Planning meeting to what plan tasks need to be done and what the goal of the Sprint is. The Team chooses tasks from the Product Backlog to complete by the end of the Sprint (Larman & Vodde, 2008). The selected Product Backlog items and the plan for how to develop and deliver these items make up the Sprint Backlog (Schwaber & Sutherland, 2017).

Daily Scrum The Daily Scrum is a 15-minute meeting held every day. The meeting should be held at the same time and place every time. According to Sutherland and Schwaber (2017, p. 12) the Daily Scrum "improve communications, eliminate other meetings, identify impediments to development for removal, highlight and promote quick decision-making and improve the Development Team's level of knowledge". Participants in Daily Scrums traditionally answer a variant of the following questions (Stray, Moe, & Sjøberg, 2020):

- What did I do yesterday?
- What are you planning to do until the next meeting?
- Are there any impediments that prevent you from making progress?

A study of 15 software teams in five countries conducted by Stray et al., 2020 suggests that these meetings should mainly focus on solving problems and planning, rather than reporting on what one did yesterday.

Sprint Review At the end of each Sprint, there is a Sprint Review where the Scrum Team and other relevant stakeholders review the Sprint (Larman & Vodde, 2008). In this meeting, the goal to inspect the increment and adapt the Product Backlog if needed (Schwaber & Sutherland, 2017).

Sprint Retrospective While the Sprint Review involves inspect and adapt regarding the product, the Sprint Retrospective involves inspect and adapt regarding the process (Schwaber & Sutherland, 2017). This meeting is held after the Sprint Review and the goal is to evaluate the process and identify what went well and what could have been done better (Schwaber & Sutherland, 2017, p. 14).

Product Backlog The Product Backlog is a sorted list of all the tasks that need to be done to complete the product. The Product Backlog provides a shared understanding of what tasks should be done and in what order (Rubin, 2012). The

Product Owner is responsible for the Product Backlog, including its contents and ordering (Schwaber & Sutherland, 2017, p. 15).

Product Backlog Grooming Product Backlog Grooming involves reviewing the Product Backlog. This can include creating and adding details to items, estimating, or prioritizing. The grooming of the Product Backlog is a continuous effort led by the Product Owner with the participation of some or all team members (Rubin, 2012).

2.1.4 Kanban

Kanban is an agile approach for development that is based on Lean principles. The principles of Lean and the principles of Kanban are largely overlapping. The five principles of Kanban is as follows (Ahmad, Markkula, & Oivo, 2013):

1. Visualise the workflow
2. Limit work in progress
3. Measure and manage flow
4. Make process policies explicit
5. Improve collaboratively

Different from Scrum, Kanban is a task-oriented process. One of the central elements of Kanban is the Kanban board, a tool for visualizing the workflow. Using a board to visualize tasks "makes it clearer for developers to understand the overall direction of work, and helps them to manage the flow" (Ahmad et al., 2013).

2.2 Coordination

There are several definitions of coordination from a variety of different fields, such as organizational theory, economics, and computer science. Ven, Delbecq, and Koenig,

1976, p. 322 define coordination as "integrating or linking together different parts of an organization to accomplish a collective set of tasks". Malone and Crowston, 1994, p. 90 introduced the following definition: "Coordination is managing dependencies between activities".

2.2.1 Theories on coordination mechanisms

Coordination can be achieved through coordination mechanisms. There are multiple theories on coordination mechanisms. Ven et al., 1976 identifies three categories of coordination mechanisms; impersonal, personal, and group. Mintzberg, 1979 defines six coordination mechanisms; mutual adjustment, direct supervision, standardization of work processes, standardization of outputs, standardization of skills, and standardization of norms. Jarzabkowski, Lê, and Feldman, 2012 introduces the term of coordinating mechanisms instead of coordination mechanisms because of it is dynamic nature. Coordinating mechanisms are defined as "dynamic social practices that are under continuous construction" (Jarzabkowski et al., 2012, p. 907).

2.2.2 Coordination strategy

Strode, Huff, Hope, and Link, 2012 defines coordination strategy as a group of coordination mechanisms that manage dependencies in a situation. The strategy has three components: synchronization, structure, and boundary spanning. Synchronization can be achieved through synchronization activities and synchronization artifacts produced during those activities. Structure is about the arrangement of, and relations between, the parts of something complex. Proximity, availability, and substitutability are all categories of coordination mechanisms with structural qualities. Lastly, the third component of coordination strategy, boundary spanning, occurs when someone within the project must interact with other organizations, or other business units,

outside of the project to achieve project goals. There are three coordination mechanisms to boundary spanning: boundary spanning activities, boundary spanning artifacts, and coordinator roles.

Strategy component	**Coordination mechanism**	**Definition**
Synchronization	Synchronisation activity	Activities that bring all of the team members together at the same time and place that promote a common understanding of the task, process, and or expertise of the other team members.
	Synchronisation artifact	An artifact generated during synchronization activities. The nature of the artifact may be visible to the whole team at a glance or largely invisible but available. An artifact 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 can 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 artifact	An artifact produced to enable coordination beyond the team and project boundaries. The nature of the artifact may be visible to the whole team at a glance or largely invisible but available. An artifact 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.

Table 2.1: Definitions of coordination strategy components (Strode, Huff, Hope, & Link, 2012)

2.2.3 Dependency taxonomy

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 artifact, a person, or a piece of information (Crowston & Osborn, 2000). Dependencies in a project can be managed well, poorly, or not at all (Strode, 2016). Understanding dependencies and understanding which dependencies are addressed by a particular practice can help practitioners to select the appropriate practice to achieve effective project coordination (Strode, 2016).

A taxonomy 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 et al., 2012. This taxonomy consists of knowledge dependencies, process dependencies, and resource dependencies. Knowledge dependencies include requirement, historical, expertise, and task allocation dependencies. Process dependencies include activity and business process dependencies. Resource dependencies include entity and technical dependencies.

Dependency		Description
Knowledge dependency	Expertise	Technical information or task information is known by only a particular person or group and this has the potential to affect project progress.
	Requirement	Requirements are a critical input to software development because they define the basic functions and qualities the software should possess. Domain knowledge (in form of a requirement) is not known and must be located or identified, and this has the potential to affect project progress.
	Task allocation	Seeing how tasks are allocated can provide useful information because each individual might at times need to know the relationship of their task to other's. 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 depenendecy	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.

Table 2.2: A description of the eight dependency types (Strode, 2016)

2.3 Teams

2.3.1 Team or Working Group

A group with a common goal is not necessarily a team. "Despite what we call them, not all 'teams' are teams. Some so-called teams are simply groups masquerading as teams because in today's world it's important to be on something called a team" (Parker, 2003, p. 1). Katzenbach and Smith, 2005, p. 165 define teams as a "small number of people with complementary skills who are committed to a common purpose, set of performance goals, and approach for which they hold themselves mutually accountable". A team requires both individual and mutual accountability (Katzenbach & Smith, 2005). A working group, on the other hand, value individual accountability and goals. This implies that the members of the working group do not take responsibility for results other than their own (Katzenbach & Smith, 2005, p. 164).

Katzenbach and Smith, 2005 also suggest a way to tell the difference between a working group and a team (see Table 2.3).

2.3.2 Autonomous teams

Autonomous teams, also known as self-organizing teams or empowered teams, are teams with a high degree of autonomy and independence. Autonomy can occur at different levels (Moe, Dingsøyr, & Dybå, 2008):

- **External autonomy** refers to the influence of the organization, like management or other individuals outside the team, on the team's activities (Moe et al., 2008).
- **Internal autonomy** refers to the degree to which all team members jointly

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 effectively indirectly by its influence on others (such as financial performance of business)	Measures performance directly by assessing collective work products
Discusses, decides, and delegates	Discusses, decides, and does real work together

Table 2.3: Difference between working groups and teams

share decision authority, rather than a centralized decision structure where one person makes all the decisions. A group may have considerable freedom to decide what group tasks to perform or how to carry it out, but individual members of the group may have very little control over their jobs (Moe et al., 2008).

- **Individual autonomy** refers to the amount of freedom and discretion an individual has in carrying out the assigned task. High autonomy is described as individuals who have few rules or procedure constraints, high control over rules and procedures, and high control over the nature and pace of work (Moe et al., 2008).

2.4 Technical Debt

The notion of technical debt focuses on the trade-offs between quality, time, and cost (Lim, Taksande, & Seaman, 2012). Avgeriou, Kruchten, Ozkaya, and Seaman, 2016 describes technical debt as "delayed tasks and immature artifacts that constitute a 'debt' because they incur extra costs in the future in the form of the increased cost of change during evolution and maintenance". Avgeriou et al., 2016 also states that technical debt can be understood as "making technical compromises that are expedient in the short term, but that creates a technical context that increases complexity and cost in the long term." Another way to understand technical debt is as a way to distinguish the gap between the current state of a system and some "ideal" state of a system. This gap includes items that are typically tracked in a software project, such as known defects and unimplemented features (Brown et al., 2010). Brown et al., 2010 identifies the following properties of technical debt:

- **Visibility**
- **Value**
- **Present value**
- **Debt accretion**
- **Environment**
- **Origin of debt**
- **Impact of debt**

2.5 Startups

Giardino, Paternoster, Unterkalmsteiner, Gorschek, and Abrahamsson, 2016 defines software startup as "those organizations focused on the creation of high-tech and

innovative products, with little or no operating history, aiming to aggressively grow their business in highly scalable markets.". Giardino et al., 2016 also states that "being a startup is usually a temporary state, where a maturing working history and market domain knowledge leads to the analysis of current working practices, thereby decreasing conditions of extreme uncertainty".

Sutton, 2000 identifies four characteristics that are typical for startups:

- **Youth and immaturity** Startups are new or relatively young companies with very little accumulated history.
- **Limited resources** Startups have limited resources. These resources can be economical, human, or physical. The first resources invested in a company are usually focused on getting the product out, promoting the product, and building strategic alliances.
- **Multiple influences** In the early stages of companies, they might be particularly sensitive to influences from various sources like investors, customers, competitors, and partners.
- **Dynamic technologies and markets** Software startups usually develop technologically innovative products and require cutting-edge development tools and techniques.

2.6 DevOps

DevOps is a concept focuses on the relationship between development and operations, and how software is managed. "it is two core principles emphasize collaboration between software development and operations, and the use of agile principles to manage deployment environments and their configurations" (Lwakatare et al., 2016). The four main dimensions of DevOps are collaboration, automation, measurement, and monitoring (Lwakatare, Kuvaja, & Oivo, 2015). it is a relatively young phenomenon

in the world of software practitioners, and it is implemented "to shorten feedback loops and the development cycle through collaboration, automation, and frequent software releases" (Lwakatare et al., 2016). Figure 2.1 shows the concepts of collaboration and the flow in DevOps.

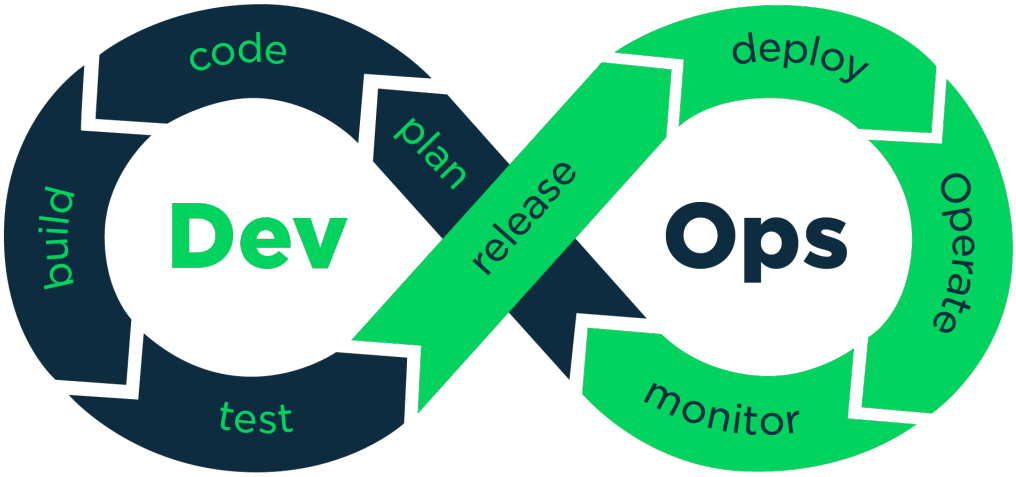

Figure 2.1: The different phases of DevOps (Harlann, 2017)

DevOps practices affect developers throughout the software development life cycle and rely heavily on tools of various kinds, such as tools for container management, continuous integration, orchestration, monitoring, deployment, and testing (Zhu, Bass, & Champlin-Scharff, 2016).

2.6.1 BizDevOps

Similarly to DevOps, BizDev tries to connect business and development to create continuous planning and integration between the two. Fitzgerald and Stol, 2017 outlines the purpose of BizDev as "The age-old disconnect between the business

strategy and technical development components is recognized in the BizDev concept which seeks to tighten this integration". Also, the role of the Product Owner is used for the connection between business strategy and development by the agile methods, but this is not necessarily sufficient (Fitzgerald & Stol, 2017). Furthermore, as an extension of BizDev, Gruhn and Schäfer, 2015 describes BizDevOps as "Business, Development and Operations work together in software development and operations, creating a consistent responsibility from business over development to operations". The concept of BizDevOps is shown in Figure 2.2.

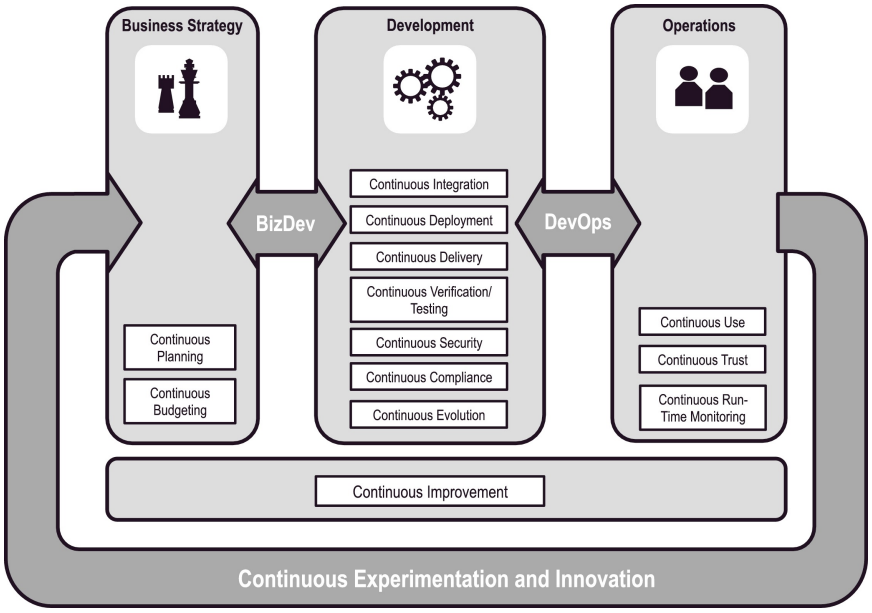

Figure 2.2: The different phases of BizDevOps (Fitzgerald & Stol, 2017)

3 | Research Method

This chapter will outline the method used in this study and why it was chosen, as well as details about the research design. First, the research method is presented. Then details on the data collection and data analysis are introduced. Lastly, the concerns regarding reliability and validity are presented.

3.1 Qualitative Research

Research can either be quantitative, qualitative, or multi-strategy (Creswell, 2018; Robson, 2011). ”Research approaches are plans and the procedures for research that span the steps from broad assumption to detailed methods of data collection, analysis and interpretation” (Creswell, 2018, p. 3). The selection of research methods should be based upon the research problem, the researcher’s personal experiences, and the audiences for the study (Creswell, 2018).

Quantitative research is an approach for "testing objective theories by examining the relations between variables" (Creswell, 2018, p. 4). Quantitative research calls for a fixed pre-specification before collecting data. The data are almost always in the form of numbers, and typical data collection methods can be surveys or experiments.

Qualitative research is an approach for "exploring and understanding the meaning of individuals or groups in the context of a social or human problem" (Creswell, 2018, p. 4). Quantitative methods rely on heavily text and image data (Creswell,

2018). A good qualitative research design generally utilizes multiple qualitative data collection techniques, such as interviews, observation, or collection documentation (Creswell, 2018; Robson, 2011). Creswell, 2018 describe the general characteristics of qualitative research as a natural setting, researcher as a key instrument, multiple sources of data, inductive and deductive data analysis, participants' meanings, emergent design, reflexivity, and holistic account.

Multi-strategy research uses a combination of qualitative and quantitative approaches in the same project. It resides somewhere in between qualitative and quantitative approaches (Creswell, 2018; Robson, 2011).

For this study, a qualitative research approach was chosen. Qualitative research methods "focus on discovering and understanding the experiences, perspectives, and thoughts of participants - that is, qualitative research explores the meaning, purpose or reality" (Harwell, 2011). As the research questions of this study involves examining how real world individuals interact with one another in their natural setting, a qualitative research approach seem to be the best fit for this study.

3.1.1 Case study

A case study is a common qualitative approach to research. "The essence of a case study, the central tendency among all types of case study, is that it tries to illuminate a decision or a set of decisions: why they were taken, how they were implemented, and with what result"(Schramm (1971), cited in Yin, 2014, p. 15). A case study is a good method for investigating a phenomenon in depth and within its real-world context (Yin, 2014).

3.2 Data Collection

The results and conclusion presented in this thesis are based upon data collected through observation, chat logs, and semi-structured interviews. Yin, 2014 presents a set of data collection principles: (1) use multiple sources of evidence, (2) create a case study database, (3) maintain the chain of evidence, and (4) exercise care when using data from electronic sources, such as chat logs (Yin, 2014).

Principle	**My approach**
Use multiple sources of evidence	The data was collected through interviews, observation, and chat logs from Slack.
Create a case study database	I kept all the collected data in a data analysis tool to build my database.
Maintain the chain of evidence	I kept all my raw data with time and place to track them easily.
Exercise care when using data from electronic sources	A healthy dose of skepticism was used when collecting data from electronic sources, trying to keep everything to the context it was meant. If possible, they were cross-checked using other sources to assess their validity.

Table 3.1: An overview of my approach to data collection principles

The data was collected in a five-month period, from **September 2018** to **March 2020**. I conducted two rounds of data collection. The first data collection was from **September 2018** to **April 2019**. The goal of this data collection was mainly to get to know the company and how they work. This data collection includes **3 interviews**

with the CTO and **1 interview with a developer (4 in total)**, as well as **1 days of observation** in their storage location. The main data collection period stretched from November 2019 to March 2020. Throughout this period, I observed a total of **31 work days**, **49 meetings**, and conducted **5 interviews** with members of the team and other relevant individuals. Table 3.2 show a timeline over the data collection.

5. September 2018	First meeting with the company.
10. September 2018	Second meeting with the company.
17. October 2018	Third meeting with the company.
20. February 2019	Fourth meeting with the company.
1. April 2019	Fifth meeting with the company.
Week 43 2019	Meeting about observation.
4. November 2019	First day of observation.
Nov 2019 - Feb 2020	Period of observation (31 days).
Nov 2019 - Feb 2020	Access to Slack.
Jan 2019 - Feb 2020	Interview period.

Table 3.2: Timeline of data collection

Method	Number	Details
Days of observation in work space	32	Study and understand the team in a natural environment. Observe processes and work flows. Conversations with team members.
Observation of meetings	49	Study and understand the team in a natural environment. Observe how meetings are conducted and how the team acts and interacts.
Interviews	9	Interviews with members of the development team.

Table 3.3: An overview of data collection from September 2018 to March 2020

3.2.1 Observation

Throughout this study, a lot of the data collected was through the use of **participant observations**. Participant observation can be understood as "a method in which a researcher takes part in the daily activities, rituals, interactions and events of a group of people as one of the means of learning the explicit and tactic aspects of their life routines and their culture" (Musante & DeWalt, 2010, p. 1). Participant observation "puts you where the action is and lets you collect data" (Musante & DeWalt, 2010, p. 2). A participant-observer has two purposes, to engage in activities appropriate to the situation and to observe the activities, people, and physical aspects of the situation (Spradley, 2016, p. 54). Participant observation differs from direct observation where the researcher does not interact with the subjects (Yin, 2014).

Through the period of this study, I observed the team in a total of **31** workdays and **49** meetings. I participated as a member of the tech team and did actual programming tasks, attended meetings and workshops, and joined the team for lunch. I was also invited to social events, such as the Christmas party. This gave me insight into how the company was structured, the different work processes in the company, the projects, and how they were executed and the work environment.

Month	**Number of days**
October 2018	1
November 2019	12
December 2019	5
January 2020	8
February 2020	6
Total	**32**

Table 3.4: An overview of work days observed

3.2.2 Interviews

Yin, 2014, p. 113 states that "Interviews are an essential source of case study evidence because most case studies are about human affairs or actions". Interviews can be either structured, semi-structured, or unstructured (Cohn, Sim, & Lee, 2009).

In this study, I conducted **9** interviews in two rounds. The first round was done from September 2018 to April 2019 and included 4 interviews, 3 with the CTO of the company, and 1 with a developer from the development team. The second round of interviews was from January 2020 to February 2020 and included 5 interviews. These interviews were mainly with the tech team or people working closely with the tech team. The main goal was to attain insight into the team's work processes, how they coordinate and prioritize tasks, how they communicate, and how they perceive

Type of meeting	**Number of meetings**
Company-wide Status Meeting	11
Daily Stand-up	19
Zendesk Planning Meeting	2
Sprint Planning	2
Backlog Grooming	1
Workshop	1
Various Tech Team Meetings	6
Various Company-wide Meetings	2
Various Project-related Meetings	2
Other	3
Total	**49**

Table 3.5: An overview of meetings observed

this themselves. It was important for me to see things from their perspective instead of mine, as I do through observations.

The interviews varied from **28** to **55** minutes with an average of **39,8** minutes. They were recorded with a tape recorder with the consent of the people interviewed. This was to provide a more accurate rendition of the interview. All interviews were then transcribed and the data was stored in the case study database. The interviews conducted were **semi-structured** and covered topics like teams and teamwork, work processes, and communication and coordination in the company. An interview guide was used to keep some structure and direction to the interviews.

Role	Gender	Time at company	When	Duration
Data Analyst	Female	4 months	January 2020	43min
Developer	Male	1,5 years	January 2020	28min
UX/UI Designer	Male	Almost 2 years	January 2020	55min
Developer	Male	1,5 years	February 2020	37min
Developer	Male	2 years	February 2020	36min

Table 3.6: An overview of second round of interviews

3.3 Data Analysis

Before presenting and discussing the results, the raw data were analyzed. The data were grouped into different groups; interviews, workplace observations and meeting observations. My analysis built on theories presented in Chapter 2 on agile software development, coordination and teams. The taxonomy proposed by Strode, 2016 was used to aid me in getting a thorough overview of the field of dependencies and coordination mechanisms. The results in Chapter 5 are therefore organized according to the taxonomy.

Furthermore, all of the data sources were uploaded into a program called NVivo[1], which is qualitative data analysis software. Descriptive coding was chosen as the coding technique which aims at summarising the topic of the selection as a word or short phrase Saldaña, 2016. This coding style is appropriate for all qualitative studies, but as it is useful for beginning qualitative researchers learning how to code data as well as studies with a wide variety of data forms Saldaña, 2016, p. 88. This is also the reason why this technique was chosen for this study.

I created a list of codes before the observations, as Miles and Huberman, 1994,

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

p. 58 recommends. During the study I revisited the codes, changing, deleting and adding codes as something new emerged or a code did not apply. Throughout the process, I kept a structure in the codes based on the main themes in the background and the dependency taxonomy by Strode, 2016. I organized my data and analyzed them based on the different coordination mechanisms and dependencies, as well as other relevant topics. I started by identifying the different coordination mechanisms, then identifying the associated category and lastly identifying the fitting dependencies. Figure 3.1 shows an example of how data were coded.

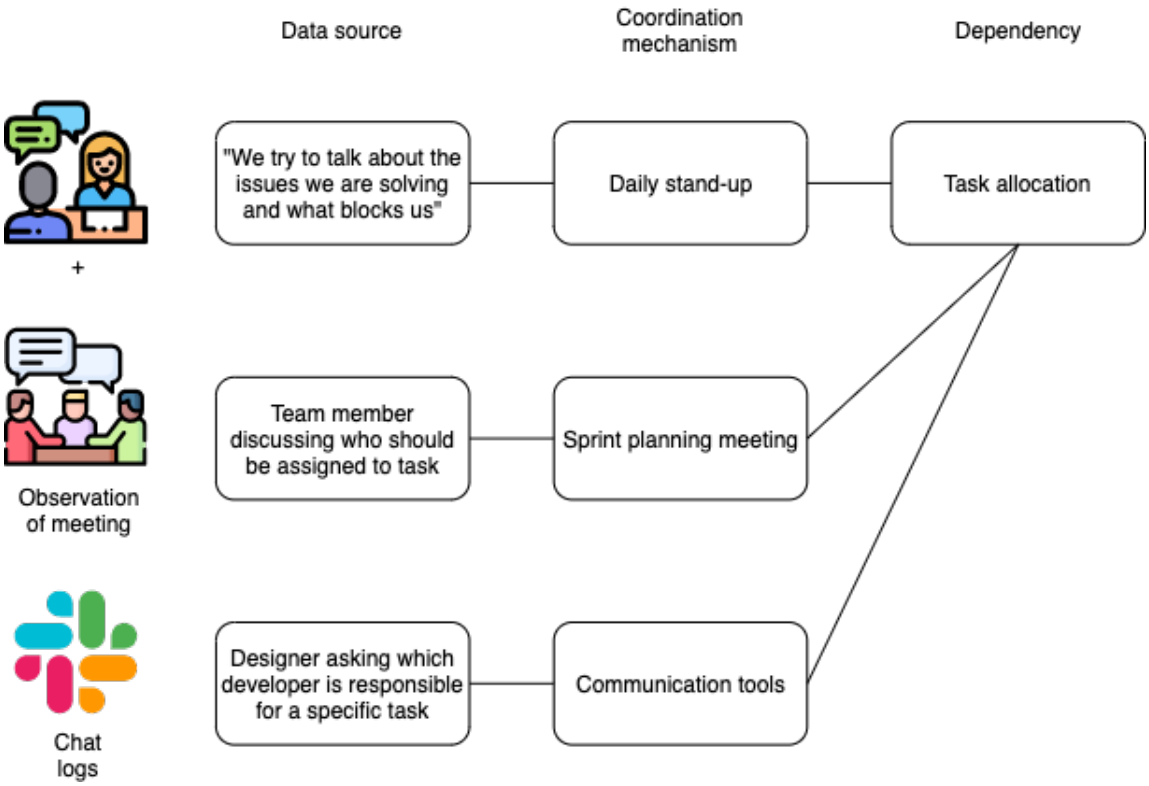

Figure 3.1: An example of how data was coded

3.4 Validity and reliability

(Yin, 2014, p. 45) presents four test for judging the quality of research design; construct validity, internal validity, external validity and reliability. The validity of a study is concerned with "the trustworthiness of the results, to what extent the results are true and not biased by the researchers' subjective point of view" (Runeson & Höst, 2009).

3.4.1 Validity

Construct validity refers to "identifying correct operational measures for the concept being studied" (Yin, 2014, p. 46). To increase construct validity in case studies, Yin, 2014 proposes to: use multiple sources of evidence, establish a chain of evidence and have the draft case study report reviewed by key informants. The first two points were used when collecting data, mentioned in chapter 3.2.

Internal validity refers to "seeking to establish a causal relationship, whereby certain conditions are believed to lead to other conditions, as distinguished from spurious relationships" (Yin, 2014, p. 46). This is mainly relevant for explanatory or casual studies, not for descriptive or exploratory studies. Since this case study is not explanatory, assessing internal validity is not relevant.

External validity refers to "defining the domain to which a study's findings can be generalized" (Yin, 2014, p. 46). Runeson and Höst, 2009 also states that this aspect of validity is concerned with to what extent the findings in a study are of relevance to other people outside the investigated case. In case of studies, Yin, 2014 suggests that the use of appropriate theory or theoretical propositions as a way to increase external validity.

3.4.2 Reliability

The reliability of a study is concerned with "demonstrating that the operations of a study - such as its data collection procedures - can be repeated, with the same result" (Yin, 2014, p. 46). The goal of reliability is to minimize errors and biases in a study (Yin, 2014, p. 49). Using a case study protocol and developing a case study database are tactics to increase reliability.

4 | Research Context

This chapter will present the research context the case study was conducted in. The organization, including the team and the work environment, is presented.

4.1 Organization

The organization investigated in this thesis is a start-up company. The company was founded in late 2016 and has around 20 employees in total. They mainly develop and maintain a service for renting out items, but also have a location to display their inventory where customers can see the products before renting.

The company is divided into management, operations, finance and development. They are located in two different locations, where the development division is seated in an office location, and management and operations are located mainly at the storage location. This does vary a lot and people move locations as they see fit. In the office location, where I mainly did my observations, there was usually 5-10 people on a daily basis.

This company was chosen for this research because of the interesting research context. The company is a fast-paced start-up working from multiple locations practicing DevOps. The company was very open to participating in this study and let me take part in all of their activities as if I was an employee.

4.2 Team

The team observed in this study was mainly the development team, consisting of 6 members: 4 developers, 1 designer and 1 data analyst. If necessary, other people assisted the development team in different projects depending on the goal of the project.

The team's task is mainly to develop and maintain the company's online service. This includes making designing, developing and testing new functionality, maintaining the system and making sure everything is working as it should at all times.

The team does not have a specific team lead for the team, but rather project leads specific to projects. They also include the role of Scrum Master, which the developers take turns in being. There is also a Product owner of each project, but this person is a person outside the development team.

It is important to decide if the observed team is a working group or a team. To answer this, I will be using the proposed theory by Katzenbach and Smith, 2005 presented in Chapter 2. Table 4.1 presents what characteristics the observed team displayed.

The observed team matches some characteristics of both a working group and a team, although the team characteristics were dominant. The team had shared leadership roles. The team would contribute to the broader organizational mission through their specific team purpose that the team itself delivered. The team had both individual work products and collective work products. They encouraged open-ended discussions and active problem-solving meetings through daily stand-ups, Zendesk meetings, project meetings and pair programming. They also ran efficient meetings.

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 effectively indirectly by its influence on others (such as financial performance of business)		Measures performance directly by assessing collective work products	✓
Discusses, decides, and delegate		Discusses, decides, and does real work together	✓
Total	**3**		**7**

Table 4.1: A working group vs. a team

The team discussed, decided and did real work together. Based on this, I would argue that the observed team can be classified as a team instead of a working group.

4.2.1 Team members

As a small team with a lot to do, their way of working was to "pitch in wherever they can". The team always focused on finishing projects and even though people have specific roles, this does not mean that they did not do things outside their area of

responsibility. In the period of observation, the designer also did a lot of marketing, and the developers also helped at the storage.

The developers shared a set of responsibilities, but some had specific responsibilities. Also, there were some tasks the developers take turns in doing, such as being Scrum Master, releasing new versions and participate in the Zendesk Planning Meetings. The main responsibility of the developers was to finish projects with everything that includes. A detailed overview of the team member's responsibilities can be found in table 4.2.

4.2.2 Seating

The office location is in a big co-working space in Oslo. The company rents a room with 6 desks and chairs where the team usually sits. The room also has a little round table and two chairs. There are a lot of other places in the building where everyone who rents space in the building can sit and work if they want.

The designer, the data analyst and one of the developers have fixed seats in the company room where they usually always sit when in the office, but for the other three developers, the seating varies a lot. Where people sit also depends on how many people from the company are working from the office location that day. The rule is mainly "first come, first served", but people rarely occupy the seats where the designer, the developer and the data analyst sits. If the company room is full, people usually choose to sit on the same floor as the company room or in the lunch area.

When observing, I usually sat on one of the desks in the middle. If the company room was full, I sat right outside the company room. Figure 4.1 displays the company room in the co-working space.

Role	Description
Data analyst	Responsible for collecting data from various sources, performing data cleaning and analysis. Does the preparation of reports to support other departments of the company. Has the main responsibility for the company's Google Analytics account setup and administration.
UX/UI Designer	Responsible for the user interface and the user experience. This includes providing support to resolve user workflow problems, develop prototypes for design ideas, design user interface and communicate design ideas to internal teams and key stakeholders. Through the period of observation, the designer was also responsible for sending out e-mails with marketing material to subscribers twice a week.
Full-stack developer and CTO	Responsible for developing the service. This includes writing and testing code, releasing updates and maintenance of the service. Has a lot of managing responsibilities.
Full-stack developer 2	Responsible for developing the service. This includes writing and testing code, releasing updates and maintenance of the service. Has the main responsibility for managing their e-mail service and managing users of the online service.
Full-stack developer 3	Responsible for developing the service. This includes writing and testing code, releasing updates and maintenance of the service.
Full-stack developer 4	Responsible for developing the service. This includes writing and testing code, releasing updates and maintenance of the service. Has the main responsibility for the system architecture and DevOps.

Table 4.2: The different roles in the team and their area of responsibilities

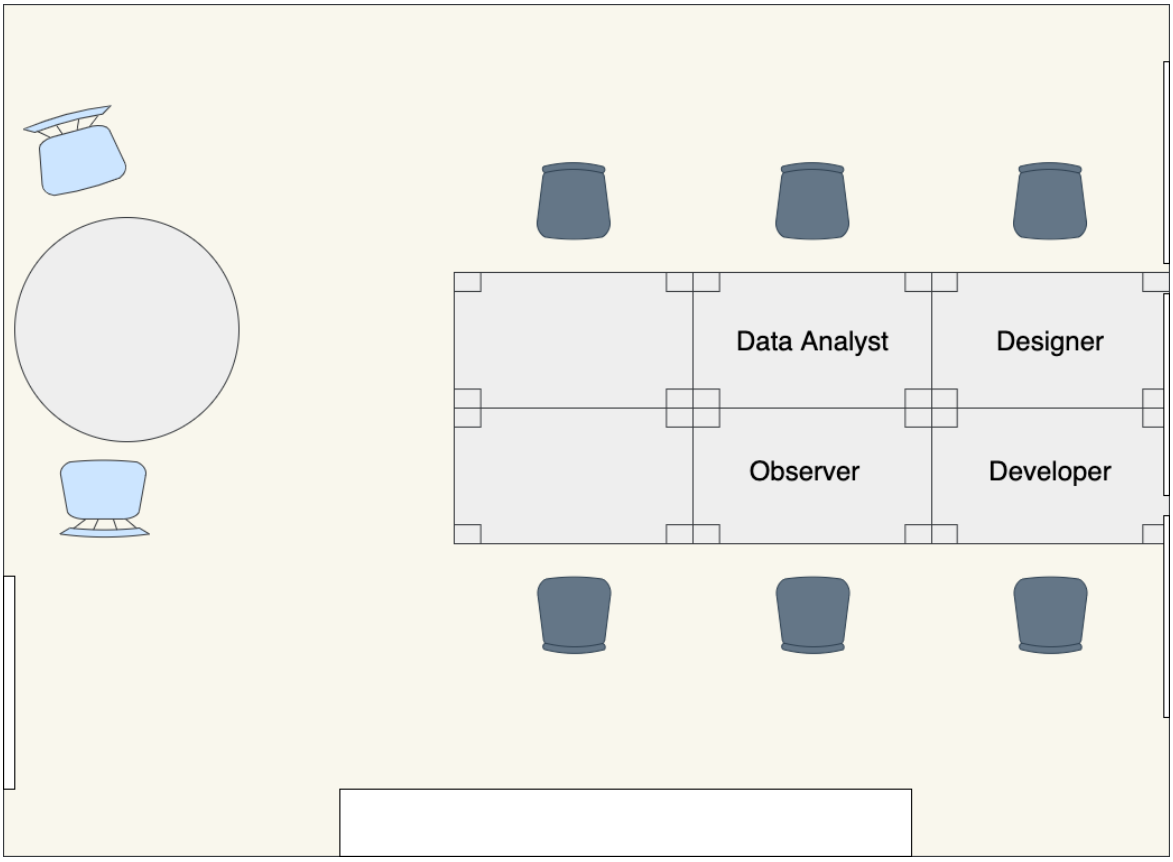

Figure 4.1: The company room

4.2.3 Tools

Zendesk[1] The company uses a customer support ticket system called Zendesk. Both customers and employees can add tickets to Zendesk. All issues, including ones from employees in the company, need to go through Zendesk. This is to have an overview of all issues in one place, to have "one source of truth", instead of having someone requesting something in an e-mail or someone asking for a change on Slack. To file requests to the development team, one has to submit it to Zendesk. Examples of requests can be to fix bugs, add discount codes, updating the design on the web page

[1]https://www.zendesk.com/

or requesting bigger updates.

Azure DevOps[2] is a service platform that provides an end-to-end DevOps toolchain for developing and deploying software. This includes Azure Boards for planning and tracking tasks, issues and projects. It also includes Azure Repos, which provides private git repos. At the start of the observation period, the development team used GitHub, but after about a month of observation, they switched to Azure DevOps because of GitHub's lack of functionality, mainly capacity planning and estimation of work. Azure DevOps also includes tools for continuous integration, monitoring and continuous deployment (Azure Pipelines). The monitoring was connected to Slack to get notifications about the status of the system. Other tools were also integrated into Azure DevOps, such as Travis CI for continuous integration.

Slack[3] is an online chat tool that allows organizations to create their own chat rooms with multiple sub-chats. Slack is the main tool of communication in the company. The company had a channel for everyone in the company, used to provide company-wide information. The developers had to channels regarding tech-related talk.

Google Meet[4] is a service for conducting video meetings. This was used for many of the company's meetings, such as the weekly company-wide status meeting and the daily stand-up. Since the company was located in multiple locations, this was a necessary tool to have their company meetings.

[2]https://azure.microsoft.com/nb-no/services/devops/

[3]Slack is a registered trademark of Slack Technologies, www.slack.com

[4]https://meet.google.com/

4.2.4 Processes

The development team follows a lean process, a light-weight version of Scrum. This was customized to fit their needs, and they involve the Scrum activities they find valuable. The following events happened regularly.

Weekly Company-Wide Status Meeting Once a week, every Monday, there was a company-wide status meeting. This was used to align all the departments in the company and update each other on the work that has been done in the previous week, and what is coming up this week. Since the company works from two different locations, the meeting was held with Google Meet and the CEO was sharing her screen with a presentation. All the departments filled in their work in the presentation before the meeting. The different departments took turns to present their work. This meeting usually took 15-30 minutes depending on how much had happened the last week.

Zendesk Planning Meeting This meeting was before the start of a new Sprint, about every two weeks, usually on Fridays. In this meeting, someone from the development team (usually developers) and someone from the other departments came together to look through the issues on Zendesk and decide what should be included in the development team's sprint.

Sprint Planning Every two weeks, the developers had a Sprint Planning meeting to plan the upcoming sprint. This usually involved breaking tasks apart, deciding who does what and agreeing upon time estimates for the tasks. It also involved capacity planning to see how much capacity there were for the next two weeks to come.

Backlog Grooming The Backlog Grooming was usually once every two weeks but was sometimes done more often if it is necessary. In these meetings, the developer team usually changed the priority of tasks, added tasks, broke issues into more

tasks, or removed tasks.

Daily Stand-up Meeting Every day, the developers had a daily stand-up. In this meeting, they gave a brief answer to the questions: 1) What did you do yesterday? 2) What are you going to do today? 3) Is there anything blocking you? This was usually done in person in the office, and if someone is working from home, they called in. If it was hard to do the stand up in person, the stand up was sometimes done in writing on Slack.

5 | Results

In chapter 5, the research context was described with an overview of the company, the departments, the roles in the development team and their work process. In this chapter, what was found through analysis of the data collected will be presented. This chapter will describe the different dependencies and their associated practices as coordination mechanisms by using a taxonomy proposed by Strode (2016). The focus will be on describing the different coordination mechanisms and dependencies.

Strode (2016) divided dependencies into three main categories; **knowledge dependency**, **process dependency** and **resource dependency**. Knowledge dependency is divided into expertise, requirement, task allocation and historical dependencies. Process dependency is split into activity and business process dependencies. Lastly, resource dependency is divided into entity and technical dependencies. Strode et al. (2012) divided coordination mechanism into eight main strategy components; **synchronization activity**, **synchronization artifact**, **boundary spanning activity**, **boundary spanning artifact**, **availability**, **proximity**, **substitutability** and **coordinator role**.

Dependency		Description
Knowledge dependency	Expertise	Technical information or task information is known by only a particular person or group and this has the potential to affect project progress.
	Requirement	Requirements are a critical input to software development because they define the basic functions and qualities the software should possess. Domain knowledge (in form of a requirement) is not known and must be located or identified, and this has the potential to affect project progress.
	Task allocation	Seeing how tasks are allocated can provide useful information because each individual might at times need to know the relationship of their task to other's. 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 depenendecy	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.

Table 5.1: A description of the eight dependency types (Strode, 2016)

5.1 Using the taxonomy to identify coordination mechanisms and dependencies

Table 5.2 shows that 95 pairs of dependencies was found in total with 38 different coordination mechanisms. Many of the practices are multipurpose and address more that a single dependency. Out of the 95 pairs of dependency pairs, 67 can be categorised as knowledge dependency, 14 can be categorised as process dependency and 14 can be categorised as resource dependency. Figure 5.1 shows the frequencies of the three main dependencies.

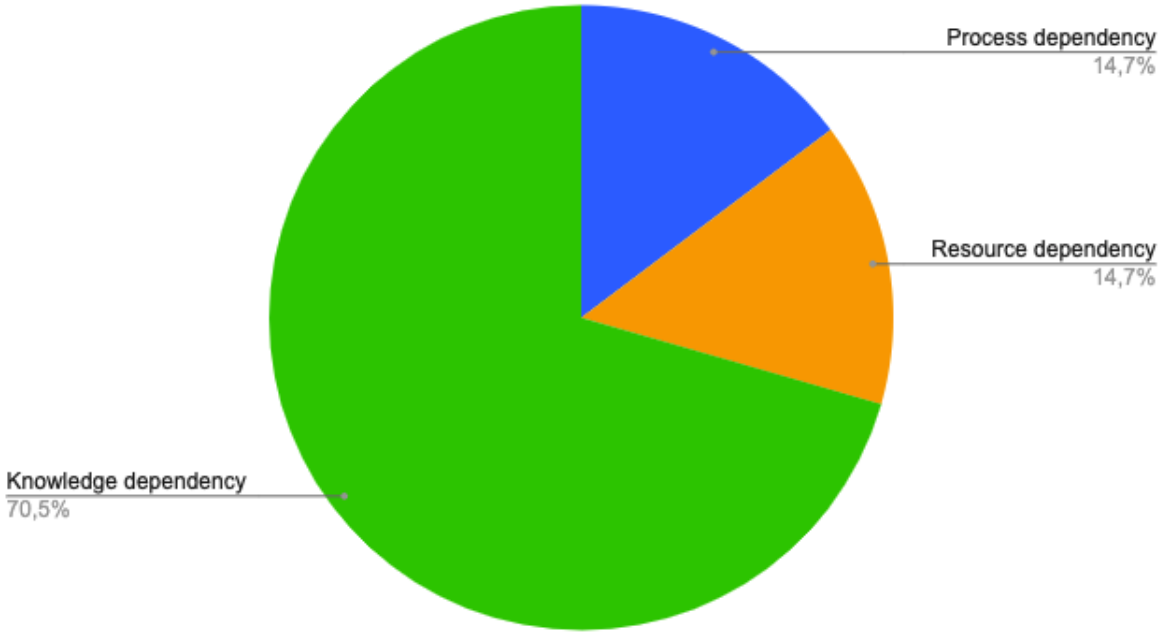

Figure 5.1: An overview of the frequency of dependencies

Coordination mechanisms (38)	Dependency								Total
	Knowledge				Process		Resource		
	Expertise	Requirement	Task allocation	Historical	Activity	Business process	Entity	Technical	
Sprint (2 weeks)		✓			✓	✓			3
Daily stand-up	✓		✓		✓		✓	✓	5
Company-wide status meeting	✓		✓						2
Ad hoc conversation	✓	✓	✓	✓				✓	5
Workshop	✓	✓							2
Retrospective	✓			✓					2
Sprint planning meeting	✓	✓	✓					✓	4
Backlog grooming	✓	✓							2
Zendesk planning meeting		✓	✓		✓	✓	✓	✓	6
Company meeting	✓		✓						2
Weekly design meeting	✓								1
Project meeting		✓	✓			✓			3
One on one meeting	✓	✓			✓	✓			4
Meeting with digital marketing agency	✓								1
Pair programming	✓								1
Software release								✓	1
Cross-team talk	✓	✓		✓		✓	✓		5
Continuous build and test								✓	1
Status meeting with supervisor	✓		✓		✓				3
Backlog		✓	✓		✓				3
Burndown chart			✓		✓				2
Wallboard	✓	✓							2
Project management tool (Azure Devops)		✓	✓						2
Wiki for development information	✓	✓							2
Task		✓	✓		✓				3
Work area	✓	✓		✓					3
Source code control				✓				✓	2
Documentation		✓		✓					2
List of prioritized tickets (Zendesk)		✓	✓						2
Communication tools (Slack, mail)	✓		✓				✓	✓	4
Pull request checklist				✓				✓	2
Test suite								✓	1
Full-time team	✓								1
Redundant skill	✓								1
Project lead	✓	✓							2
Product owner	✓	✓							2
Scrum master	✓	✓	✓						3
UX/UI Designer	✓	✓				✓			3
Total pairs of dependency	**24**	**21**	**15**	**7**	**8**	**6**	**4**	**10**	
	67				**14**		**14**		**95**

Table 5.2: Identified coordination mechanisms in the development team

Practice	Total of dependencies	Best matched dependency
Zendesk planning meeting	6	Process dependency
Daily stand-up	5	Knowledge dependency
Ad hoc conversation	5	Knowledge dependency
Cross-team talk	5	Knowledge dependency
One on one meeting	4	Knowledge dependency
Sprint planning meeting	4	Knowledge dependency
Communication tools	4	Resource dependency
Status meeting with supervisor	3	Knowledge dependency
Project meeting	3	Knowledge dependency
Backlog	3	Knowledge dependency
Task	3	Process dependency
Work area	3	Knowledge dependency
Scrum master	3	Process dependency
Sprint	3	Process dependency
UX/UI Designer	3	Knowledge dependency

Table 5.3: Agile practices found to address three or more dependencies

5.2 Dependencies and Coordination Mechanisms

To find the practices that promote a smooth workflow, Table 5.3 shows the practices that were found to address three or more dependencies and therefore helps to ensure a smooth workflow in the project. These practices are matched to the best-fitted dependencies.

Table 5.4 shows both dependencies and coordination mechanisms to illustrate how coordination mechanisms address dependencies. The coordination mechanisms are also mapped to the strategy component to follow the taxonomy. The 38 identified practices are listed to strategy components and mapped to the dependencies they manage. A practice can address more than a single dependency.

The coordination mechanisms presented in Table 5.5 are presented in more depth in section 6.3, 6.4 and 6.5. These mechanisms are selected because of the substantial data collection on them and because they are of good use to the team. The coordination mechanisms that address more than one dependency is presented within their best-matched dependency.

Table 5.6 shows the frequency of the different regular meetings working as coordination mechanisms.

			Dependency							
			Knowledge				Process		Resource	
			Expertise	Requirement	Task allocation	Historical	Activity	Business process	Entity	Technical
Coordination Mechanisms	Syncronization activities	Daily stand-up	■		■		■		■	■
		Company-wide status meetings	■		■					
		Status meeting with supervisor	■		■		■			
		Workshop	■	■						
		Sprint		■			■	■		
		Retrospective	■			■				
		Zendesk planning meeting		■	■		■	■	■	■
		Backlog grooming	■	■						
		Sprint planning meeting	■	■	■					■
		Company meeting	■		■					
		One on one meeting	■	■			■	■		
		Pair programming	■							
		Cross-team talk	■	■		■		■	■	
		Continuous build and test								■
		Software release								■
		Weekly design meeting	■							
		Project meeting		■	■			■		
	Syncronization artifact	Backlog		■	■		■			
		Burndown chart			■		■			
		Wallboard	■	■						
		Project management tool		■	■					
		Wiki	■	■						
		Documentation		■		■				
		List of prioritized tickets		■	■					
		Pull request checklist				■				■
		Task		■	■		■			
		Communication tools	■		■				■	■
		Source code control				■				■
	Bondary spanning activities	Ad hoc conversations	■	■	■	■				■
		Meeting with digital marketing agency	■							
	Boundary spanning artifacts	Test suite								■
	Availability	Full-time team	■							
	Proximity	Work area	■	■		■				
	Substitutability	Redundant skill	■							
	Coordinator role	Project lead	■	■						
		Product owner	■	■						
		Scrum master	■	■	■					
		UX/UI Designer	■	■				■		

Table 5.4: Dependencies and coordination mechanisms identified in the development team

Dependency		**Coordination Mechanisms**
Knowledge dependency	Expertise	Cross-team talk Workshop Wallboard Work area
	Requirement	Ad hoc conversations Designer Backlog grooming
	Task allocation	Daily stand-up meeting Sprint planning meeting Company-wide status meeting
	Historical	Documentation Source code control
Process dependency	Activity	Sprint
	Business process	Zendesk planning meeting
Resource dependency	Entity	Communication tools
	Technical	Software release

Table 5.5: The selected coordination mechanisms that will be described

Coordination mechanism	**Frequency**
Daily stand-up	Daily
Company-wide status meetings	Weekly
Status meeting with supervisor	Weekly
Workshop	Every other week
Retrospective	Rare
Zendesk planning meeting	Every other week
Backlog grooming	Every other week
Sprint planning meeting	Every other week
Project meeting	Every other week
Weekly design meeting	Weekly

Table 5.6: Frequency of the regular events

5.3 Knowledge dependency

The company had many coordination mechanisms regarding knowledge dependencies. I found that 35 out of 38 could be categorized as a knowledge dependency. In table 5.7 all of the coordination mechanisms managing the knowledge dependencies are presented.

Expertise (24)	Requirement (21)	Task allocation (15)	Historical (7)
Daily stand-up	Workshop	Daily stand-up	Retrospective
Company-wide status meeting	Sprint	Company-wide status meeting	Cross-team talk
Status meeting with supervisor	Zendesk planning meeting	Status meeting with supervisor	Documentation
Workshop	Backlog grooming	Zendesk planning meeting	Pull request checklist
Retrospective	Sprint planning meeting	Sprint planning meeting	Source code control
Backlog grooming	One on one meeting	Company meeting	Ad hoc conversations
Sprint planning meeting	Cross-team talk	Project meeting	Work area
Company meeting	Project meeting	Backlog	
One on one meeting	Backlog	Burndown chart	
Pair programming	Project management tool	Project management tool	
Cross-team talk	Wallboard	List of prioritized tickets	
Weekly design meeting	Wiki	Task	
Wallboard	Documentation	Communication tools	
Wiki	List of prioritized tickets	Scrum master	
Communication tools	Task	Ad hoc conversations	
Ad hoc conversations	Ad hoc conversations		
Meeting with digital marketing agency	Project lead		
Full-time team	Product owner		
Redundant skill	Scrum master		
Work area	UX/UI designer		
Scrum master	Work area		
Project lead			
UX/UI designer			
Product owner			

Table 5.7: Coordination mechanisms managing the knowledge dependency

5.3.1 Expertise

Cross-team talk

The work area made it easy for different departments to talk to each other. There was often people from the different departments in the company's workroom. This led to cross-team talks, often happening multiple times a day. Often, task information was known by people outside the development team and cross-team talk helped the development team gain the knowledge needed to solve their tasks. These types of conversations often helped to manage expertise dependencies, but also requirement dependencies, historical dependencies, business process dependencies, and entity dependencies.

Workshop

The developers held workshops or "tech talks" about different topics related to software development. This usually involved one developer presenting the topic for the other developers and having some kind of related practical tasks. The main goal of the workshop was to share knowledge among the team members. In the period of observation, a developer held a workshop about a layout system called CSS Grid where he held a short presentation about the topic and demonstrated it's functionality. This was something that could be relevant to the project.

Wallboard

Wallboards were placed in all meeting rooms and company rooms. These wallboards were often used to demonstrate some concept or discuss solutions and ideas to both people in the development team as well as people from other departments.

Work area

The development team often sat in the office space with people from other departments. The work area facilitated coordination through easy access to both team members and people from other departments. The area enabled frequent discussions

and conversations about tasks, requirements and possible solutions. Right outside the office space were meeting rooms available and more open work areas which contributed to informal and unscheduled meetings. The work area included wallboard which was frequently used.

5.3.2 Requirement

Ad hoc conversations

Ad hoc conversation happened frequently, often several times a day, in the work area and other places in the building where the team members were located. The work area made it easy for the team members to have conversations, ask each other questions or have informal discussions. These conversations helped the team make quick decisions, discuss tasks and requirements with the relevant people, and get a good sense of what the other people were working on. Ad hoc conversations were most often observed in the company's workroom, but also sometimes other places in the building, depending on where people were sitting.

Designer

The designer usually sat in the company's workroom. This made it easy for the developers to show their work to the designer to check if it fit the requirements or ask questions related to the tasks at hand. On some projects, the designer acted as the project owner. In the interview with the designer, it became clear that he does a lot of coordination for the team and spent a lot of time gathering information from the other departments.

The designer stated that: *I would say I work with almost everybody at [company], maybe not daily, but collecting information comes not from the team I just mentioned [development team], but also from the showroom, so going there a couple of times monthly is very important because I need to know what the customers are*

saying.

Backlog grooming

On one of the days of observation, three members of the development team had an ad hoc backlog grooming session to understand the requirement of the project better and break it down into smaller tasks. In this session, there were also assigned people to different tasks and someone responsible for the different issues getting done in the course of the sprint.

5.3.3 Task Allocation

Daily stand-up meeting

The daily stand-up was important for managing task allocation dependencies and expertise dependencies. The developers in the team had their stand up at 10:15 every morning. The reason for having it at 10:15 was because everyone had usually come in to work at this time. The meeting was usually carried out in their office space next to their seats. If someone was not at the office yet, they called in on Google Meet on someone's computer. The meeting started with someone stating what they did yesterday, what they are going to do today, and if something was blocking them. If any of the developers had met any obstacles, other team members contributed with valuable input on how to solve their problems or who to talk to to get it done. The discussion usually ended up with coordinating tasks and deciding who should be involved.

A developer stated that: *"I mean, at first I didn't really like it, but it gives you a good idea of what the team as a whole is working towards, and it's good to know what the other people are working on in case what you are working on might affect their work, so you can collaborate and come up with some solution."*

Some of the problems brought up in the daily stand-ups were also tackled in later meetings if they didn't have any clear or good solution at the time. After everyone in the team had answered all three questions, they also provided information which could be relevant for the others, like the outcome of a meeting or a change in the agenda.

One of the developers stated that this meeting could be more focused on problem-solving and that: *"I feel like it can be improved in that sense for us because it just feels like you're reporting and it's not meant to be for that.*

Sprint planning meeting

The sprint planning meeting occurred every two weeks. This usually happened at 10:15 on Mondays and all the developers have set aside 1 hour for it (the duration varies). The purpose of this meeting was to plan the upcoming Sprint. In this meeting, the developers started by taking a look at the tasks from the last sprint to see if there were any unfinished tasks. Unfinished tasks from the last week, tasks from the product backlog and tasks from the Zendesk meeting was then broken apart into smaller pieces. They also included technical debt if they had the capacity. The developers then coordinated who did what and made time estimations for each of the tasks. They tried to write down the information needed on each task, and if some information was missing, they discussed who to talk to to get the information needed to finish the task.

Company-wide status meeting

Every Monday at 12:00 the whole company had a weekly status meeting. These meetings typically lasted 15 to 30 minutes. All employees in the company participated in these meetings. All the departments updated the rest of the company on what they did the last week and what they were going to do in the upcoming weeks. Every department head, and sometimes different project leaders, gave a brief update

on what the department was working on and who was involved in what.

One developer stated that: *"We have these weekly company-wide meetings where we hear from all the different departments and know what's going on. That's really helpful because you feel more motivated when you see that everything is in place for us to work together and get to a certain goal."*

5.3.4 Historical

Documentation

The company kept a Google Drive for all the documents in the company. This included all presentations, plans, project descriptions, and so on. All employees had access to this and this made it easy for the employees to look back at the documentation of earlier projects or plans to gain knowledge about past decisions.

Source code control

The developers used Git as a source code control system (version control). Based on my observations, this made it a lot easier to collaborate on code and track changes done to the code. If the developers wondered about something done in the past, it was easy to find it in earlier versions of the code.

5.4 Process dependency

I found that 11 out of 38 coordination mechanisms could be categorized as process dependency. In table 5.8 all of the coordination mechanisms managing the process dependencies are shown.

Activity (8)	Business process (6)
Daily stand-up	Sprint
Status meeting with supervisor	Zendesk planning meeting
Sprint	One on one meeting
Sprint planning meeting	Cross-team talk
One on one meeting	Project meeting
Backlog	UX/UI designer
Burndown chart	
Communication tools	

Table 5.8: Coordination mechanisms managing the process dependency

5.4.1 Activity

Sprint

The sprints in the development team were conducted over two weeks. Before the sprint even started, one or more representatives from the development team had been at the Zendesk planning meeting to align the upcoming tasks and projects with the relevant departments. At the start of a Sprint, the developers conducted Sprint planning meetings. The team conducted backlog grooming when they felt the need to refine tasks and requirements. Retrospectives were conducted whenever the team felt the need to reflect upon their process. At the end of the sprint, a new software release was done and the company was updated on the progress done the last to weeks.

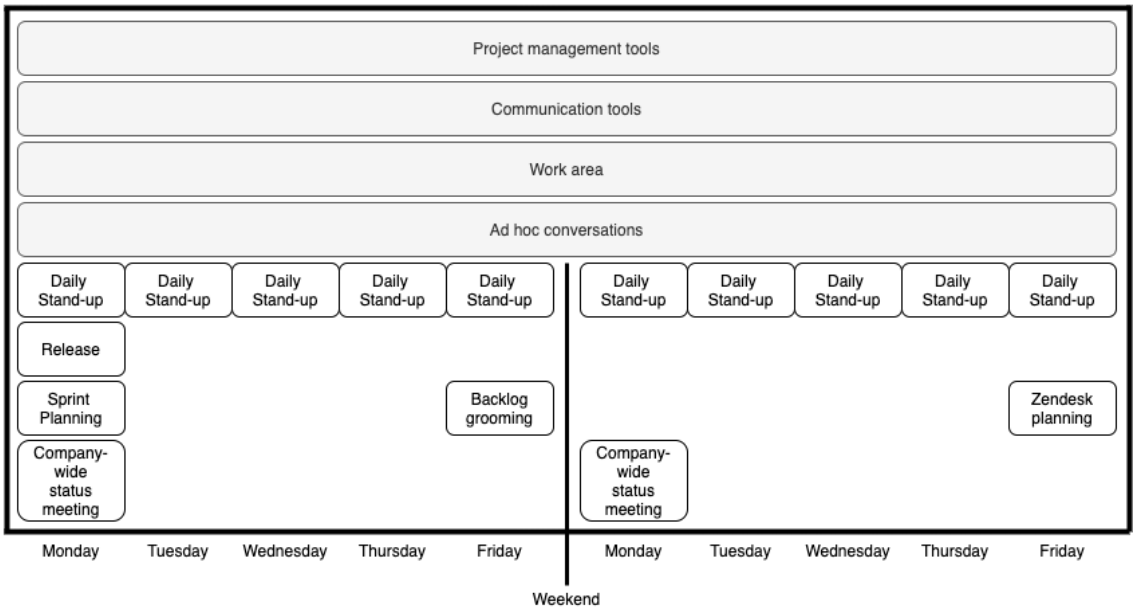

Figure 5.2: An overview over the coordination mechanisms present in a Sprint

5.4.2 Business process

Zendesk planning meeting

This meeting was usually held on Fridays at 11:00 every two weeks, before the beginning of a new Sprint. In this meeting, representatives from the development team and the other departments in the company met to prioritize and choose tasks for the upcoming sprint. This included looking through the list of tickets (can be bugs, issues, new functionality) on Zendesk and picking out a few for the development team to tackle. The tasks chosen in the Zendesk meeting was usually tasks wanted done by other departments (such as discount codes) or bugs issued by customers using the website. This meeting helped align the development team's plans with the plans of the other departments.

The designer stated that: *"Zendesk has helped us a lot with the process. It's the one place and everyone has to create issues with the site no matter what, and they put it in there and then we go through it. I think that has been really good for us that on Fridays we always go through this list, we close tickets, we open tickets and then we pick the new ones for the next week."*

5.5 Resource dependency

I found that 11 out of 38 coordination mechanisms could be categorized as managing resource dependencies. In table 5.9 all of the coordination mechanisms managing the resource dependencies are presented.

Entity (4)	Technical (10)
Daily stand-up	Daily stand-up
Zendesk planning meeting	Zendesk planning meeting
Cross-team talk	Sprint planning meeting
Communication tools	Continuous build and test
	Software release
	Pull request checklist
	Communication tools
	Source code control
	Ad hoc conversation
	Test suite

Table 5.9: Coordination mechanisms managing the resource dependency

5.5.1 Entity

Communication tools

The team made use of several communication tools, such as Skype, Google Meet and e-mail. Slack was used for communications with different parts of the company through channels. Slack was used to inform others about issues, new releases and other work-related tasks. Since the employees often worked from different places, Slack was frequently used to contact people. They sometimes also sent each other messages on Slack when they are in the same room instead of interrupting each other

in person. It was also used for more social reasons, such as inviting others to lunch or inform about social events.

One of the developers stated that: *"It's a little special because the company is in two different locations, but we take full advantage of Slack so it is easy to contact people."*

Another developer stated that: *" I usually communicate via Slack because I'm not going to run around trying to find people. If it's urgent, I will track them down, but it's usually always on Slack. Its easier to just write them if I need something and then I have it documented somewhere."*

Google Meet was used to have video or audio conversations. This happened every Monday in the Company-wide status meetings since the employees sat in different locations and sometimes also in other meetings.

5.5.2 Technical

Software release

The developers took turns in doing the software release. When doing a software release, the designated person from the development team had to follow a checklist to make sure the new changes were ready to be released. When a new release had been done, they made an update to the tech team in the tech team channel on Slack and an update to everyone in the company on their company-wide Slack channel.

5.6 Barriers for managing dependencies

In the period of the data collection, a few barriers were identified. A lot of the things in the company, such as roles, teams, and processes felt very fluid and not "set in stone". This could sometimes contribute to issues due to things being unclear, confusing and uncertain.

5.6.1 Role clarity

A barrier within the company was the role clarity. As a small company, all employees had to pitch in wherever needed. This meant that sometimes people had to help in the storage location if someone were ill or do other things which were not usually their responsibility.

For some of the employees, their role or the role of others was not clear. A few felt like they did not know what to do or what they were responsible for. When I asked the data scientist what her role at the company was, her reply was: *"I would like to know. At least a little bit more precisely, but in general I'm supposed to help with the data. Whatever that implies."*.

The roles of product owner, scrum master and project lead were also not very clear. Everyone had some understanding of what it implied, but it was still not clear what the specific tasks and responsibilities of the roles were. What the tasks of the different roles were depended on who had the role.

5.6.2 Working remotely

From time to time, usually once or twice every week, people worked from other locations than the company's locations. Some thought this was very helpful to the team and made it easier to focus on getting tasks done. A developer stated that: *"I try to work from home, work remotely, at least one time a week. I think that is really good because if you're in a position where you feel like you need to finish a lot of work, then being in a different environment or just being by yourself sometimes, you can just hammer through and just finish what you want. I mean, sometimes, when you're at the office, there might be like some small spontaneous meeting or a small discussion and these can be a distraction. So yeah, I think its good for each of us to try to do that at least one time a week or so".*

Other people on the team meant that the company was not ready for remote work yet and that it would be better if everyone was in the same location. One of the developers stated that: *"I don't think the company is ready for remote work because we are a small team that needs to move fast. We need to be aligned with each other fast. I have seen companies that actually are working on for example a legacy application that has to be maintained and it's very possible that someone is working from Brazil and is maintaining a part of the application, but here is not like that, so it's always frustrating when people are working remotely. It's harder to get people together and they need to call in and there is always a lot of extra work."*

The designer had some experience with remote work and stated that: *"I do think that working remotely is fine, but I don't usually do it myself. I think it's better to be focused at work and then go home and be offline. I did work remote full-time during my last year of university with my last job and that was intense because you never see the people. That is one of the main reasons I wanted to work here and work in smaller teams again because I needed a break from that kind of remote work. If you're not careful I think it can be kind of demotivating because you're not around people*

that can help with ideas more efficiently. I would say remote work is totally normal and sometimes even helpful to focus without interruptions."

5.6.3 Integrating the data analyst

One of the challenges in the company was efficiently integrating the data analyst. The data analyst had little experience with the work when she got hired and since she was the only one working with the data, it was hard for her to get the needed guidance and help. The data analyst stated that: *"There is literally nobody I can ask. And I know if I was working in a company that actually deals with those things or there is a bigger team and there would be somebody with more experienced that I could ask. If I want to actually ask somebody with competence, that would be a consulting company and that's like 1600 NOK per hour, so if I want to ask a question, that is gonna cost."* This made it hard for the data analyst to work effectively because she had to deal with most of her issues on her own. If there had been anyone with more experience in her field, it would have been easy for her to ask questions or discuss solutions.

5.6.4 Planning and estimation

Early in the observation period, in a chat with a member of the operations department, a need for better planning and estimations from the development team was wished upon. It was hard for other departments to understand what the development was going to do in the near future and how much time it was going to take. This would be nice to understand in order to know when other departments could expect their requests to be fulfilled by the development team.

The designer stated that: *"It's always very difficult to predict timelines with the developers. Some project takes longer, some projects take less time, so you just never know. Also, some people are more developed in how they can predict, others are not. "*

The designer's work often depended on the work of the developers. The designer stated that: *"It sometimes frustrating for me because that [the developer's work] would also mean how much time I have to work on something they need at a certain time, but if they are working on another project and that makes their current project longer, then that means that I actually had more time and then I feel like I should not have gone so fast and I would have been able to think about ideas and spent more time on choosing an idea. If you have more time, you can do more research. Also, if I'm working on the next design project and the developers are two projects behind, that is not really a good way of working either, because things change and the company too, in that amount of time that those projects could be implemented. It's not always easy, but we are getting better with the process."*

To make it easier for the developers to know exactly how much time they had to work on actual development tasks, the development team introduced "meeting-free days". This involved that they could only be called into meetings on Mondays or Fridays. Meetings could happen other days, but these needed to be of a high priority. Having Tuesday, Wednesday and Thursday to just focus on work helped the team to finish their tasks and be able to focus on a task for a longer period without distractions.

The designer stated that: *"We try to do meetings on Monday and Friday. Mondays and Fridays are meeting days and the rest of the week is focused on getting work done. It's easier to just think more clearly when you know which days you have to get things done and it's easier to prioritize. It also allows more organic conversations on Tuesday to Thursday if one needs to check-in or something. So yeah, I would say it makes things easier."*

A developer stated that: *"You can just call me to meetings on Monday or Friday*

because I need dedicated time to focus on stuff."

This seemed to make a great impact on the developer's focus and made it easier for them to know how much work they could take on in a given week because they knew how many working days they had dedicated to actual work (and not scheduled meetings).

Problems related to planning and estimation also got a bit better due to the implementation of light-weight Scrum. This made it easier for everyone to know what was supposed to be done in the upcoming two weeks. Also, the tasks had a time estimate attached to them, making it easier to get an overview of how the time would be spent.

5.6.5 Implementing changes in the software development process

Through the interviews and the period of observation, it became very clear that the development team valued their freedom to work however they wanted and from wherever they wanted. One thing I noticed in my period of observation, was that it was hard to change processes and standards or implement new processes and standards. When someone purposed to change something, everyone was open to listening to the proposition, but the person who suggested the change had to have a really good reason for the change. Suggestions for changes in processes often needed a lot of discussions and it was hard work getting everyone on board with it because everyone had to agree with it. At one point, in a meeting about the development team's processes, one of the developers stated that: *"Our processes are all over the place! We need a process and it's better with an OK process than no process at all."*

One of the developers also stated that: *"I think, like having all the bright people*

with sometimes very different opinions, it can be difficult to find common ground. Everyone has their view on how things should be done and it can be difficult to have a space to agree on something and move forward on this. So we need more listening and collaboration to make sure that happens. I think maybe we... it's an asset that we have this creativity and freedom and being able to do what we think is right, but at the same time putting some structures to it... It feels like a lot of resistance whenever there is a new process or a new standardization or adopting something a little more rigid - there is a lot of push back. So, I think this is good, but it makes it challenging to set up processes."

This also got a little better when agreeing to stick to light-weight Scrum for at least a few months to give it a chance and see if it could be useful to the team.

6 | Discussion

This section will discuss the results presented in Chapter 5. First, the results and relevant theory will be used to discuss the research questions. Then, implications for practice and theory will be discussed, followed by the limitations of this study.

6.1 Dependencies and their associated practices

The developers sometimes had to wait for information from particular employees or departments to finish their work. Sometimes they had to wait for a specific activity to be done or wait for domain knowledge to do their tasks.

This section will discuss the following research question:
RQ1: How are dependencies managed in an agile DevOps company?

My case study provides evidence for all eight types of dependencies, including expertise, task allocation, requirement, historical, activity, business process, entity and technical dependencies. These dependencies were identified using the dependency taxonomy by Strode (2016). My findings show that the aggregated category of knowledge dependencies accounts for 70,5% of all dependencies, whilst both process dependencies and resource dependencies account for 14,7% each. This is relevant because it indicates that addressing knowledge dependencies should have a vital impact on coordination in agile DevOps companies. The findings also show that 15

multi-purpose practices are addressing three or more dependencies.

The company had both scheduled meetings, such as the scrum ceremonies and the company-wide status meeting, and unscheduled meetings, such as ad hoc conversations, to manage dependencies within the company. The scheduled meetings usually happen Monday or Friday due to the developer's need for time to focus on their work tasks Tuesdays, Wednesdays, and Thursdays. Moe, Dingsøyr, and Rolland, 2018 recommend to identify the important scheduled meetings early as having enough scheduled meetings is important to develop a common understanding of domain knowledge. Berntzen, Moe, and Stray, 2019 found that a focus on achieving high-quality communication changes coordination over time and that unscheduled coordination enables high-quality coordination. Eisenbart, Garbuio, Mascia, and Morandi, 2016 found that scheduled meetings with a pre-distributed agenda led to more personal conflict, while unscheduled meetings led to unbiased discussions of task-related conflicts.

The team needed two types of planning meetings, the Zendesk planning meeting, and the Sprint planning meeting. The Sprint planning managed dependencies within the team, while the Zendesk planning meeting handled dependencies from outside the team. It is also a difference in what kind of dependencies they managed.

6.1.1 Knowledge dependency

The results presented in the previous chapter showed that 70,5% of the practices identified act as coordination mechanisms for knowledge dependencies. Some of the most important knowledge dependency managing coordination mechanisms includes the daily stand-up, the company-wide status meeting, the sprint planning meeting and ad hoc conversations. All of these coordination mechanisms manage 2 or more knowledge dependencies. Ad hoc conversations manage all of the 4 knowledge de-

pendencies, making it one of the crucial coordination mechanisms.

Daily stand-up

The goal of the daily stand-up was to gain insight into the team members' tasks and tackle any obstacles in the way of making progress. The stand-up was vital for managing task allocation dependencies and expertise dependencies.

A study done by Stray et al., 2020 where data was collected over 8 years including interviews with 60 project members with roles at all levels in 15 different teams shows that it is difficult to implement stand-ups in a way that benefit the whole team. One of the main problems found in this study was that the information shared in the daily stand-up was not perceived as relevant, particularly due to the diversity in roles, tasks and seniority. In my study, this was not a problem, because most of the time, only the developers took part in the daily stand-up. They were often working on a small number of tasks and many of the tasks were connected in some way, so the information given in the stand-up was almost always relevant for most of the participants.

Another problem with the daily stand-up is that managers or Scrum Masters use the meeting primarily to receive status information. Out of the time spent on the Three Scrum Questions in this study, 66% was spent on the "Have Done" question (Stray et al., 2020). In my study, the developers sometimes felt that they just reported what they were working on (this meaning the "Have Done" and "Will Do" questions). They also expressed that problem-focused communication was the most important part of the daily stand-up because it helped them identify and solve problems and make decisions more effectively. Stray et al., 2020 suggests to stop asking "What did you do yesterday?". This is to reduce time spent on status reporting and self-justification, focus on future work and spend time discussing and solving problems as well as making quick decisions.

In my study, the team held their meetings at 10:15 every day. Sometimes team members had not yet arrived at the office, trying to call into the meeting whilst on the bus or in traffic, making it hard for the rest of the team to hear what they were saying. This often ended in that the meeting was delayed by 10-15 minutes. The timing of the meeting did often disturbed the focus of the team and sometimes the meeting was held off until someone was done with a task. Stray et al., 2020 suggests finding the least disruptive time to hold the stand-up. A suggestion is to have the meeting before lunch to decrease the number of distractions.

Company-wide status meeting

The weekly company-wide status meeting made sure all employees knew what was going on in the company. It was really important for managing expertise and task allocation dependencies.

Sprint planning meeting

The goal of the Sprint planning meeting was to break issues down to smaller tasks and delegate them to the team. This meeting also included estimating time on each task, discuss requirements and talk about who to get in touch with if more information was needed. The meeting lasted for about an hour.

Abrahamsson, Salo, Ronkainen, and Warsta, 2002 recommend that the Sprint planning meeting should be a two-phase meeting organized by the Scrum master. Abrahamsson et al., 2002 suggests that in the first phase of the meeting, the customers, users, management, Product Owner and Scrum Team participate to decide upon the goals and the functionality of the next Sprint. The second phase of the meeting should be held by the Scrum Master and the Scrum Team focusing on how the product increment is implemented during the Sprint. Stray, Moe, and Aasheim, 2019 found that in their study, the two phases had the opposite order. First, 60 min-

utes with pre-planning was conducted to plan unfinished tasks from the last Sprint and plan new tasks for the upcoming Sprint. After this pre-planning, a 60-minute Sprint planning was held where the goal was to agree on the tasks list suggested in the pre-planning. This differs from what I found in my study.

In my study, the Zendesk meetings acted as a pre-planning meeting for the upcoming Sprint, but it did not involve the whole team. It usually involved representatives from other departments such as operations, finance or marketing. The Sprint planning was held after the Zendesk meeting only involving the developers. This is more similar to what Abrahamsson et al., 2002 suggests, except for that the pre-planning (the Zendesk meeting) and the actual Sprint planning were two different meetings on two different days.

Ad hoc conversations

Ad hoc conversations managed all of the four knowledge dependencies. Ad hoc conversations happened daily and took place everywhere, especially where the team where located. The work area made it easy for the team members and employees from other departments of the company to have quick discussions. Ad hoc conversations often happed within the development team, but also often with people outside of the team. The developers often had discussions about standards and how to solve problems, but also a lot of conversations with the designer about requirements. The casual communication about unclear tasks or requirements often leads to unscheduled meetings.

In a study done by Stray, 2018 on planned and unplanned meetings in large-scale projects, it was found that the employees spent more time in ad hoc conversations and unscheduled meetings than they did in scheduled meetings. The study also suggests that ad hoc conversations provide an important venue for coordination as the discussion in unscheduled meetings and ad hoc conversations may be more focused

and lead to more effective decision making than discussions in scheduled meetings. This is also true for my study.

6.1.2 Process dependency

The results presented in the previous chapter showed that 14,7% of the practices identified act as coordination mechanisms for managing process dependencies. Both the Zendesk planning meeting and the Sprint manages both activity and business process dependencies.

Sprint

The development team had a Sprint period of two weeks. When I first started the observations, the Sprints for the development team was only a week due to all other departments having one-week sprints. This caused a lot of frustration amongst the developers because it was not enough to complete projects. Two weeks fit the development team better and made it easier to take on more tasks or even finish projects. It did, however, make it more complicated to coordinate with the other departments because the other departments didn't a clear plan for more than one week. Also, the Zendesk meeting used to be once a week when the development team had weekly sprints, but after changing to two-week periods, the Zendesk meeting only happened every other week. This meant that the other departments had to plan more precisely to get their requests done in time.

The Sprint period worked as a coordination mechanism managing both activity dependencies and business process dependencies. This is because a Sprint is the main period for completing activities before other activities and business process can proceed.

Zendesk planning meeting

Then Zendesk planning meeting worked as a pre-planning meeting for the upcoming sprint and was the most important meeting to allocate tasks and requirements based on the requests from other departments. It also helped align the development team's work with the work of other departments. In these meetings, it was

easy for the other departments to voice their needs, and inform the development team about upcoming events. In these meetings, the Zendesk tickets issued by the other departments were prioritized so the development team knew what needed to be done first. An example of a task that was allocated in this meeting was a discount code for the participants of an upcoming event that the marketing team was going to.

6.1.3 Resource dependency

The results presented in the previous chapter showed that 14,7% of the practices were classified as coordination mechanisms managing the resource dependencies. The daily stand-up, the Zendesk planning meeting, and communication tools all manage both entity and technical dependencies.

Communication tools

Communication tools such as Slack, e-mail and Google Meet were frequently used to coordinate and collaborate. Communication tools contributed to managing entity dependencies by being able to reach out to people who were not present. As the company was distributed, it was crucial to be able to connect with people in other locations.

A study done by Stray, Moe, and Noroozi, 2019 on the use of Slack in virtual agile teams showed that positive aspects of using the tool were increased transparency, team awareness, and informal communication. Slack also facilitates problem-focused communication which is essential for agile teams. In my study, a lot of informal communication happened on Slack. It was also used to discuss problems and make decisions.

Communication tools managed entity and technical dependencies, but also knowledge dependencies and process dependencies, making conversation tools a crucial coordination mechanism in the company.

6.1.4 My findings compared to other studies

Table 6.1 shows a comparison of the coordination mechanisms addressing three or more dependencies in my study, in a study done by Stray, Moe, and Aasheim, 2019 and in a study done by Strode, 2016. Both my study and the study done by Stray, Moe, and Aasheim, 2019 includes daily stand-up, sprint planning meeting, communication tools, ad hoc conversations and work area. Stray, Moe, and Aasheim, 2019 also found that Scrum of Scrum meetings and team leader meetings addressed three or more dependencies. These meetings were not present in my study, but in the study done by Stray, Moe, and Aasheim, 2019 due to the study being done in an large-scale context. My study and the study done by Strode, 2016 also has some common coordination mechanisms, such as cross-team talk, sprints, iteration planning session (or sprint planning meeting) and backlog.

My study	Stray et al. (2019)	Strode (2016)
Zendesk planning meeting		Cross-team talk
Daily stand-up		Informal face-to-face negotiation
Ad hoc conversation		with external parties
Cross-team talk	Scrum of Scrum meeting	Sprints of 1 to 2 weeks
One on one meeting	Team leader meeting	Wallboard displaying current
Sprint planning meeting	Daily stand-up	stories, tasks, and task assignment
Communication tools	Sprint planning meeting	User stories
Status meeting with supervisor	Ad hoc conversations	A co-located team
Project meeting	Communication tools	Iteration planning session
Backlog	Project management tools	Story breakdown sessions
Task	Kanban board	A product backlog
Work area	Open work area	A done checklist
Scrum master		Working software at the
Sprint		end of each sprint
UX/UI Designer		A single priority team

Table 6.1: Comparison of coordination mechanisms which addresses three or more dependencies

Change over time

Jarzabkowski et al., 2012 argued that coordinating mechanisms do not appear as ready-to-use techniques but are formed as actors go about the process of coordinating. Furthermore, coordinating mechanisms are not stable entities but emerge through their use in ongoing interactions (Jarzabkowski et al., 2012). This is consistent with the findings in this study. Many of the coordination mechanisms in this study change under the period of observation. At the point of observation start, none of the Scrum-related activities were a part of the team's practices. Also, some of the practices were changed and improved for example through the retrospective.

6.2 Barriers for managing dependencies

This section answering and discuss the second research question:
RQ2: What are barriers for managing dependencies in an agile DevOps company?

There were some barriers to managing dependencies in the company which now will be discussed. Barriers such as having to work remotely, lack of role clarity, integrating the data analyst, planning and estimation, and implementing changes in the software development process. All of these barriers made it harder to manage dependencies.

Working remotely
Working remotely made managing several dependencies harder due to people not being physically at the work location. This could sometimes make it hard managing dependencies, especially if the person working remotely were not available through communication tools.

Role clarity
When the roles are ambiguous it could be hard to locate who knows what or who does what. Some of the employees have tasks that are outside of what they would regard as their responsibilities. This made them stressed and frustrated, not knowing what could be expected of them. The data analyst clearly stated that she did not know what was her responsibilities and not. She knew she was responsible for helping the data but was not sure what that implied.

The product owner is responsible for maximizing the value of the product resulting from the work of the development team (Schwaber & Sutherland, 2017, p.6). Berntzen et al., 2019 states that a core responsibility of the product owner is to communicate business needs to the development team. In their study, they found that

coordination varies depending on the context of the product owner. In my study, I found that the role of the product owner was unclear and did not have a set of defined tasks and responsibilities. The product owner participated in project meetings to communicate business needs and project requirements but did not participate in sprint planning, backlog grooming or daily stand-ups. Rubin, 2012 states that the product owner should act as a link between the Scrum Team and the stakeholder. Hence, the product owner must understand the needs and priorities of the stakeholders, the customers, and the users to represent them in the development and ensure that the right product is developed. To represent the stakeholders and users, I would argue that it would be beneficial if the product owner was more involved in choosing tasks for the Sprint and continually re-prioritizing and refining the list (Larman & Vodde, 2008).

In my study, the role of Scrum Master had more defined tasks and responsibilities than what the product owner had. The role of Scrum Master rotated between the developers, so what the Scrum Master did depended on who was the Scrum Master at the time. The Scrum Master always facilitated the Sprint planning and shielded the team from outside interference. The Scrum Master was also responsible for making sure that they followed the process of Scrum. This includes facilitating the stand-up, but in practice, whoever remembered the meeting where the one making sure it happened. Table 6.2 shows the findings from my study compared to the findings from a study by Bass, 2014 and a study by Dingsøyr, Moe, and Seim, 2018.

Integrating the data analyst role

Another challenge in the team was to integrate the data analyst which is closely related to role clarity. The team got a data analyst in October of 2019, so they did not know how to incorporate her to the team in an efficient way. Before, data analysis and other data-related tasks were done by the developers, but because they did not have the time for it, the company decided to hire a data analyst. Since neither the

This study	Bass, 2014	Dingsøyr, Moe, and Seim, 2018
Stand-up facilitator Process anchor Sprint planner Developer Impediment remover	Process anchor Stand-up facilitator Impediment remover Sprint planner Scrum of Scrum facilitator Integration anchor	Stand-up facilitator Iteration planner Demonstration facilitator Retrospective facilitator

Table 6.2: The functions of a Scrum master in my study, Bass, 2014 and Dingsøyr, Moe, and Seim, 2018

data analyst nor the team knew what her role was exactly, it made it hard to know how to utilize this role in a good way.

A study done by Hukkelberg and Berntzen, 2019 identified a set of challenges related to integrating data science roles in agile autonomous teams. Hukkelberg and Berntzen, 2019 found that misconceptions about the data scientist role lead to wrong expectations about what a team wants the data scientist to solve, preventing the realization of the full value of having this role on the team. This fits well with what I found in my study. The role of the data analyst was unclear to most of the team and the other departments, making it hard to know what the data analyst could solve. This made it hard to utilize the value of this role. Another challenge related to integrating data science roles can be to enable collaboration and knowledge sharing (Hukkelberg & Berntzen, 2019). For data science roles to grow and learn, it is important to use the time working with other people in data science roles. The only data-related role in the company I studied was the data analyst. It was difficult for her to grow in her role at the high pace that was needed because there was no-one there for her to ask if she met any obstacles. From time to time, the developers could help with the technical aspect, but not the data analyst part of her work. In the interviews, she voiced a need for people in the same role to talk to, ask questions,

and discuss problems with.

To integrate the data analyst, both the team and company must understand the role of a data analyst and what a data analyst can and should contribute with (Hukkelberg & Berntzen, 2019). This also makes it easier for the data analyst to know what her role in the team is and what the others can expect of her. It would also be good for the data analyst to have someone to collaborate and share knowledge with, so I would suggest looking into other data-related roles, such as data scientist or data engineer to support her. This depends on the needs of the company, but it can at least be worth looking into to make the most of the data the company collects.

Planning and estimation

Planning and estimation are important for managing task allocation dependencies, activity dependencies, and business process dependencies. A problem in the company was the lack of clearly expressing how long the developers thought a project or a task would take. This made it hard for the designer to know how long he had to do his tasks. It also made it hard for the operations department to plan when projects should be done in the future as some activities and business processes could not be carried out without the finished work of the developers or the development team.

Implementing changes in the software development process

The team often resisted implementing changes in the software development process. This could make it harder to manage dependencies because coordination needs change over time. If one does not add, change or remove coordination mechanisms based upon the needs of the team, it can be hard to manage dependencies well because the coordination mechanisms in the team may not be fitted to the need of the team.

6.3 Implications for theory

The dependency taxonomy by Strode (2016) was used in this study to identify dependencies and coordination mechanisms. I found that this taxonomy worked well in my case. It was a valuable help to identify the different dependencies in the company. I also felt that the different dependencies described by Strode fit well with my results and there were no dependencies that I identified that did not fit the description of one of the dependencies identified in the taxonomy. The taxonomy was also applicable when identifying coordination mechanisms. The taxonomy also made it possible to match coordination mechanisms to the best fit strategy components and dependencies.

6.4 Implications for practice

Firstly, companies should continuously evaluate their coordination needs. The need for coordination changes over time, just as companies and projects change over time. When teams introduce new coordination mechanisms or change existing ones, it is important to consider which dependencies should be managed. I would suggest prioritizing coordination mechanisms managing a high number of dependencies.

A great strength of the company, in this case, was their frequent use of communication tools. Since the company was located in two different locations and a lot of people often worked from home, cafes or other countries, this helped the team get the information they needed to do their tasks. Also, the development team often messaged each other on Slack instead of talking to each other face to face to not disturb people when they were focusing on something. This helped to keep focus and answer whenever the person had time for it.

An interesting finding in this study was the need for two different planning meetings. This was done because the development team did not only need to plan their sprints, but they needed to align with business outside of their own team, such as marketing, finance or operations. I would argue that it is important to have planning meetings that do not only involve the development team but the stakeholders outside of the team as well.

Lastly, I would also suggest making sure everyone in the team have the same understanding of what it means to have the role of Scrum Master, Product Owner, and Project Lead. It is important that these roles are clear and that people taking on these roles know what they are supposed to do and what responsibilities they have.

6.5 Limitations

The limitations of this study include those common to qualitative case study research in general and those specific to this study.

Case study

Choosing a different case in a different context might have led to other different coordination mechanisms and dependencies being identified.

Development team's point of view

This study mainly observed coordination from the development team's point of view. The observations were done mainly in the office space where the development team sat and not in the storage area where the business part of the company was located. The result of this study could have been different if I had spent more time observing and interviewing the business department in the company.

Taxonomy by Strode (2016)

The dependency taxonomy by Strode (2016) was used to identify dependencies and coordination mechanisms in this study. The results presented by Strode (2016) are based on three different cases. The cases varied in complexity and the projects had different degrees of involvement of the customer. Moreover, the results by Strode (2016) are based upon interviews with the team members, while in my study the results are based upon interviews and extensive observations. Therefore, it is important to keep in mind the difference in scale and type when comparing my study to Strode (2016).

Coordination over time

Since "coordination mechanisms are dynamic social practices that are under continuous construction" (Jarzabkowski et al., 2012) it is evident that the results could be

different if the data was collected at a different point in time. Therefore, it could have been interesting to do more observation of the team over a longer period.

Other frameworks

There are other frameworks that could have been chosen, such as the theory proposed by Van De Ven et al. (1976). If a different framework had been used, the results and discussion could have been different.

Construct validity

As mentioned in Chapter 3.2 on data collection I followed the principles presented by Yin, 2014. I used multiple sources of evidence in the form of interviews, observations, documentation, and chat logs. This helps ensure the validity of the research.

External validity

The use of a theory is a way to increase the external validity of a study. The dependency taxonomy for agile software development projects by Strode (2016) was used in this study.

Reliability

To increase the reliability of the study, I created a case study database and maintained a chain of evidence, as mentioned in Chapter 3.2.

7 | Conclusion and further work

In this thesis, I have presented a case study of an agile DevOps company. I have presented a theoretical background on the relevant topics to understand the company and its processes. Also, I presented the research methods used, the research context and results from the study. Lastly, the results were discussed in light of relevant research. This chapter will provide a conclusion of the study and suggestions for future work.

The first research question aimed at understanding how dependencies were managed in an agile DevOps company. I found a total of 38 practices acting as coordination mechanisms. A total of 15 coordination mechanisms addressing three or more dependencies were identified. The most crucial coordination mechanisms included Zendesk planning meeting, the daily stand-up, ad hoc conversations, sprint planning meeting and communication tools. All of these mechanisms addressed four or more dependencies and happened regularly. They also managed dependencies across the different categories of dependencies.

The second question aimed to identify and discuss barriers for managing dependencies efficiently in an agile DevOps context. The company were facing challenges related to working remotely, role clarity, planning and estimation, and implementing changes in the software development process.

7.1 Future work

Several topics could be interesting to research further.

The need for coordination and coordination mechanisms change over time. Therefore, it could be interesting to study how they change over a longer period. It could also be interesting to study how and why new coordination mechanisms are introduced.

As communication tools were one of the most crucial coordination mechanisms in this study. This helped the team make quick decisions, discuss topics and solve problems. It could be interesting to analyze chat logs to get a better understanding of what dependencies are managed by communication tools and how.

In this study, some of the coordinator roles were unclear. Therefore, it could be interesting to study coordinator roles such as Product Owner, Scrum Master and Project lead to understand how these roles act as coordination mechanisms, which dependencies they manage and how they manage these dependencies.

Lastly, the development team in this study had dedicated days for meetings. It could be interesting to research this topic further to understand the benefits and challenges of this practice, and how this affects the coordination in the team.

References

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods: Review and analysis. *Proc. Espoo 2002*, 3–107.

Ahmad, M. O., Markkula, J., & Oivo, M. (2013). Kanban in software development: A systematic literature review, 9–16. doi:10.1109/SEAA.2013.28

Avgeriou, P., Kruchten, P., Ozkaya, I., & Seaman, C. (2016). Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162). *Dagstuhl Reports*, *6*(4), 110–138. doi:10.4230/DagRep.6.4.110

Bass, J. M. (2014). Scrum master activities: Process tailoring in large enterprise projects. In *2014 ieee 9th international conference on global software engineering* (pp. 6–15). IEEE.

Berntzen, M., Moe, N., & Stray, V. (2019). The product owner in large-scale agile: An empirical study through the lens of relational coordination theory. (pp. 121–136). doi:10.1007/978-3-030-19034-7_8

Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., ... Zazworka, N. (2010). Managing technical debt in software-reliant systems. In *Proceedings of the fse/sdp workshop on future of software engineering research* (pp. 47–52). FoSER '10. ACM.

Cohn, M., Sim, S., & Lee, C. (2009). What counts as software process? negotiating the boundary of software work through artifacts and conversation. *Computer Supported Cooperative Work*, *18*, 401–443. doi:10.1007/s10606-009-9100-4

Creswell, J. W. (2018). Research design : Qualitative, quantitative mixed methods approaches. Los Angeles, California: Sage.

Crowston, K., & Osborn, C. (2000). A coordination theory approach to process description and redesign. *Former Departments, Centers, Institutes and Projects*.

Dingsøyr, T., Moe, N. B., & Seim, E. A. (2018). Coordinating knowledge work in multiteam programs: Findings from a large-scale agile development program. *Project Management Journal*, *49*(6), 64–77.

Ebert, C., Abrahamsson, P., & Oza, N. (2012). Lean software development. *IEEE Software*, *29*(5), 22–25.

Eisenbart, B., Garbuio, M., Mascia, D., & Morandi, F. (2016). Does scheduling matter? when unscheduled decision making results in more effective meetings. *Journal of Strategy and Management*, *9*, 15–38. doi:10.1108/JSMA-03-2014-0017

Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, *123*, 176–189. doi:https://doi.org/10.1016/j.jss.2015.06.063

Giardino, C., Paternoster, N., Unterkalmsteiner, M., Gorschek, T., & Abrahamsson, P. (2016). Software development in startup companies: The greenfield startup model. *IEEE Transactions on Software Engineering*, *42*(6), 585–604.

Gruhn, V., & Schäfer, C. (2015). Bizdevops: Because devops is not the end of the story. In H. Fujita & G. Guizzi (Eds.), *Intelligent software methodologies, tools and techniques* (pp. 388–398). Cham: Springer International Publishing.

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

Harwell, M. R. (2011). Research design in qualitative/quantitative/mixed methods. In C. F. C. R. C. Serlin (Ed.), *The sage handbook for research in education: Pursuing ideas as the keystone of exemplary inquiry* (Chap. 10, pp. 147–164). doi:10.4135/9781483351377

Hemon, A., Fitzgerald, B., Lyonnet, B., & Rowe, F. (2020). Innovative practices for knowledge sharing in large-scale devops. *IEEE Software*, *37*(3), 30–37.

Hukkelberg, I., & Berntzen, M. (2019). Exploring the challenges of integrating data science roles in agile autonomous teams. In R. Hoda (Ed.), *Agile processes in software engineering and extreme programming – workshops* (pp. 37–45). Cham: Springer International Publishing.

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. doi:10.1287/orsc.1110.0693

Katzenbach, J. R., & Smith, D. K. (2005). The discipline of teams. *Harvard Business Review*, *83*(7/8), 162–171.

Larman, C., & Vodde, B. (2008). *Scaling lean & agile development: Thinking and organizational tools for large-scale scrum*. Addison-Wesley Professional.

Lim, E., Taksande, N., & Seaman, C. (2012). A balancing act: What software practitioners have to say about technical debt. *IEEE Software*, *29*(6), 22–27.

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? In *2016 49th hawaii international conference on system sciences (hicss)* (Vol. 2016-, pp. 5437–5446). IEEE.

Lwakatare, L. E., Kuvaja, P., & Oivo, M. (2015). Dimensions of devops. In C. Lassenius, T. Dingsøyr, & M. Paasivaara (Eds.), *Agile processes in software engineering and extreme programming* (pp. 212–217). Springer International Publishing.

Malone, T., & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*, *26*(1), 87–119.

Miles, M., & Huberman, A. (1994). Qualitative data analysis: An expanded sourcebook. Thousand Oaks, Calif: Sage.

Mintzberg, H. (1979). The structuring of organizations : A synthesis of the research. Englewood Cliffs, N.J: Prentice-Hall.

Moe, N. B., Dingsøyr, T., & Dybå, T. (2008). Understanding self-organizing teams in agile software development. In *19th australian conference on software engineering (aswec 2008)* (pp. 76–85). IEEE.

Moe, N. B., Dingsøyr, T., & Rolland, K. (2018). To schedule or not to schedule? an investigation of meetings as an inter-team coordination mechanism in large-scale agile software development. *6*, 45–59. doi:10.12821/ijispm060303

Musante, K., & DeWalt, B. (2010). *Participant observation: A guide for fieldworkers*. AltaMira Press. Retrieved from https://books.google.no/books?id=ymJJUkR7s3UC

Nerur, S., & Balijepally, V. (2007). Theoretical reflections on agile development methodologies - the traditional goal of optimization and control is making way for learning and innovation. *Communications Of The Acm*, *50*(3), 79–83.

Parker, G. (2003). *Cross-functional teams; working with allies, enemies, and other strangers*. Wiley.

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Agile software development series. Addison-Wesley.

Raman, S. (1998). Lean software development: Is it feasible? In *17th dasc. aiaa/ieee/sae. digital avionics systems conference. proceedings (cat. no.98ch36267)* (Vol. 1, C13/1–C13/8 vol.1). IEEE.

Robson, C. (2011). Real world research : A resource for users of social research methods in applied settings. Chichester: Wiley.

Rubin, K. S. (2012). *Essential scrum: A practical guide to the most popular agile process*. Addison-Wesley Professional.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164.

Saldaña, J. (2016). The coding manual for qualitative researchers. London: Sage.

Schwaber, K., & Sutherland, J. (2017). The scrum guide. the definitive guide to scrum: The rules of the game. Retrieved from https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pd

Spradley, J. (2016). *Participant observation*. Holt, Rinehart and Winston. Retrieved from https://books.google.no/books?id=q7DlCwAAQBAJ

Stray, V. (2018). Planned and unplanned meetings in large-scale projects. (pp. 1–5). doi:10.1145/3234152.3234178

Stray, V., Moe, B. N., & Aasheim, A. (2019). Dependency management in large-scale agile: A case study of devops teams. In *Proceedings of the 52nd hawaii international conference on system sciences*.

Stray, V., Moe, N. B., & Noroozi, M. (2019). Slack me if you can! using enterprise social networking tools in virtual agile teams. In *2019 acm/ieee 14th international conference on global software engineering (icgse)* (pp. 111–121).

Stray, V., Moe, N. B., & Sjøberg, D. (2020). Daily stand-up meetings: Start breaking the rules. *IEEE Software*, *37*(3), 70–77.

Strode, D. (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. *The Journal of Systems Software*, *85*(6), 1222–1238.

Sutton, J., Stanley M. (2000). The role of process in a software start-up. *IEEE Software*, *17*(4), 33.

Ven, A. H. V. D., Delbecq, A. L., & Koenig, R. (1976). Determinants of coordination modes within organizations. *American Sociological Review*, *41*(2), 322–338.

Yin, R. K. (2014). Case study research : Design and methods. Los Angeles, Calif: SAGE.

Zhu, L., Bass, L., & Champlin-Scharff, G. (2016). Devops and its practices. *IEEE Software*, *33*(3), 32–34.

Appendices

A Observation protocol

Observation protocol.

Topic	Question
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?

B Interview guide

Interview Guide

Introduction

- Present myself and say a little about the project
- Thank the person for participating
- Confirm confidentiality and anonymity
- Ask for permission to record the interview

General

- How long have you been working for this company?
- What role do you have? What are your tasks and responsibilities?
- What in general are you working on now?

Team

- Who do you consider a part of your team?
- Do you have an overview of what the other team members are doing?
- Do you feel like you (and your team) have a common goal?
- Do you collaborate with other team members? How and on what typically? Is it useful?

Processes

- How are tasks chosen and prioritized?
    - Who decides and how?
- Could you talk me through the process from you get a task until its finished?
- Do you (and your team) track or have an overview of the tasks you are working on? How?
    - If yes - is it useful?
    - Is there any prioritization of the tasks?
- Do you have any regular events? (Ex. weekly meetings, daily meetings, weekly deployment)
    - How do you feel about these events? Are they useful?
- Is there anything about the processes in the team that could be better? Do you have any thoughts on how?
- Has anything about this process changed over the time you have worked here?

Coordination

- How are problems that emerge in the project solved?
- Who do you talk to in order to solve your tasks?
- What do you normally need to clarify with others?
- How easy is it to continue on others' work?
- What do you spend a lot of time on? And not?
- How much time do you spend in meetings, and on [development tasks/design-related tasks]? How do you feel about this?
- How often are meetings moved or canceled? What are usually the reasons for canceling or moving a meeting?
- How do you coordinate with your team or people outside your team?
- How do you feel about the flow of information, both in your team and in the company?
- How often are you working from the office and how often are you working from home? What do you prefer (if you prefer one over the other and what situations)? Why?

Closing

- What is the best thing about working here?
- What is the worst or most frustrating thing about your job?
- How is working here different from other jobs you have had?
- Is there anything you would like to add that we did not discuss?
- Thank you for participating!