NTNU
Norwegian University of Science and Technology

Method Tailoring in Agile Software Development Projects

An Exploratory Case Study

Lars Kråkevik

Master of Science in Engineering and ICT
Submission date: February 2018
Supervisor: Nils Olsson, MTP
Co-supervisor: Torgeir Dingsøyr, IDI

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

Preface

This thesis concludes my five year master's program in Engineering and ICT at the Norwegian University of Science and Technology (NTNU). The master thesis was conducted as an exploratory case study in the course Project and Quality Management (TPK4920) over a period of one semester, and have been a very enlightening and rewarding process.

Throughout the study program I have taken a broad selection of courses. In several of these, agile software development have been a prominent theme that caught my interest. I further pursued this in a specialization project prior to this thesis, which also introduced me to the concept of method tailoring. I found this to be a very intriguing and highly relevant topic in the context of agile software development projects that have yet to be adequately investigated in research. This motivated to further explore this in my master thesis.

First and foremost, I would like to give special thanks to my two supervisors for creating this opportunity to pursue my interest for agile software development. A big thank you to my main supervisor Nils Olsson for helpful guidance, feedback, and continued support throughout. Thank you also to my co-supervisor Torgeir Dingsøyr for the many helpful guidance sessions, and for sharing your expertise on agile software development and related topics.

I would also like to thank the companies and its representatives who gave me the opportunity to study a very interesting project and use it as a case in my thesis.

Lastly, a thank you to my family for all the support and helpful advice all the way from start to finish.

Trondheim, February 14, 2018

Lars Kråkevik

Lars Kråkevik

Summary

Information technology is becoming increasingly important for organizations and individuals in today's society. Software is a key part of this and can provide competitive edge in a market characterized by constant change. This makes it challenging for those who develop and deliver these products and services. To succeed from idea, via development, testing and execution, an agile project method that enable quick handling of change is essential.

Agile methods have typically recommended delivery of change and new features in increments every two to four weeks. In recent times, a trend is observed towards agile approaches that enable delivery on a continuous basis for additional benefits and competitiveness. However, this requires companies to effectively tailor agile methods and apply mature technology with aim towards this in their projects and deliveries.

Method tailoring is a very relevant issue in conjunction to this, which involves tailoring of project methods to the actual needs and goals of a project context. This thesis was conducted as a case study which main purpose was to explore a software development project's approach to method tailoring, and how an agile method was tailored in such a manner that deliveries of software changes and new features could be done several times a day. The main findings showed that ongoing tailoring was a necessary activity for continuous improvement of the method in the project. A prominent characteristic of the tailoring approach was that the project suppliers did not confine to a method framework, but were flexible and selected agile practices from multiple methods based on needs and experiences. This entailed a combination of agile practices into a minimal method by omitting unnecessary practices and formalities. The tailored method was based on a flow-based development model rather than an iterative model, and consisted of several agile practices that together enabled the suppliers to deliver continuously, promptly and with the right priorities.

Keywords: Agile methods and practices, method tailoring, continuous software delivery, project management

Sammendrag

Informasjonsteknologi blir stadig viktigere for organisasjoner og enkeltindivider i dagens samfunn. Programvare er en sentral del av dette, og kan gi konkurransefortrinn i et marked som er preget av konstant endring. Dette gjør det ekstra krevende for de som utvikler og leverer disse produktene og tjenestene. Skal man lykkes fra idé, via utvikling, test og produksjonsetting er en smidig prosjektmetodikk som raskt håndterer endringer vesentlig.

Smidige metoder har typisk foreslått leveranser av endring og ny funksjonalitet i inkrementer hver andre til hver fjerde uke. I nyere tid ser man en trend mot smidige tilnærminger som muliggjør leveranse på en kontinuerlig basis for ytterligere fordeler og konkurranseevne. Dette krever imidlertid at selskaper på en effektiv måte tilpasser smidige metoder og tar i bruk moden teknologi med sikte mot dette i sine prosjekter og leveranser.

Metodetilpasning er et særdeles relevant tema i denne forbindelse, som tar for seg tilpasning av prosjektmetoder til de faktiske behov og mål i en prosjektkontekst. Denne avhandlingen ble gjennomført som et case studie hvor formålet var å utforske et programvareutviklingsprosjekts tilnærming til metodetilpasning, samt hvordan en smidig metode ble tilpasset slik at leveranser av endringer og ny funksjonalitet kunne gjøres flere ganger om dagen. Hovedfunnene viste at fortløpende tilpasninger var sentralt for kontinuerlig forbedring av metoden i prosjektet. En fremtredende karakteristikk for tilnærmingen var at man ikke begrenset seg til et metoderammeverk, men var fleksible og valgte smidige praksiser fra flere metoder etter behov og erfaringer. Dette innebar en kombinasjon av smidige praksiser til en minimal metode ved å utelate unødvendige praksiser og formaliteter. Den tilpassede metoden tok utgangspunkt i en flytbasert utviklingsmodell fremfor en iterativ modell, og besto av flere smidige praksiser som til sammen gjorde det mulig for leverandørene å levere kontinuerlig, raskt og med de rette prioriteringer.

Nøkkelord: Smidige metoder og praksiser, metodetilpasning, kontinuerlige leveranser, prosjektledelse

Contents

1 Introduction 1

1.1 Background and Motivation 1

1.2 Problem Description 4

1.2.1 Clarifications and Definitions 5

1.3 Scope 6

1.4 Target Audience 7

1.5 Thesis Structure 8

2 Background Literature and Theory 11

2.1 Agile Software Development 11

2.1.1 Traditional versus Agile Software Development 11

2.1.2 Agile Principles and Agility 14

2.1.3 Towards Continuous Software Delivery in Agile Software Development 17

2.2 Agile Methods and Practices 18

2.2.1 Agile Methods Overview 18

2.2.2 Iterative Methods 19

2.2.3 Flow-based Methods 24

2.2.4 Agile Practices 29

2.3 Method Tailoring 31

2.3.1 Introduction to Method Tailoring Theory 31

2.3.2 Method Tailoring Approaches 34

2.4 Summary 39

3 Method 41

3.1 Literature Review 41

3.1.1 Literature Inclusion and Exclusion Strategy 42

3.2 Case Study Research 44

3.3 Case Selection 46

3.4 Preparation for Data Collection 47

3.5 Data Collection 49

3.5.1 Semi-Structured Interviews 49

3.5.2 Observations and Document Analysis 50
3.6 Transcription of Recorded Interviews 51
3.7 Coding and Data Analysis 52
3.8 Reporting of Results 54
3.9 Confidentiality and Anonymization 54
3.10 Research validity, reliability and generalizability 55

4 Results 57
4.1 Overview of the Project 57
4.2 The Use and Tailoring of Agile Methods and Practices 61
4.2.1 Approach to Agile Method Tailoring 62
4.2.2 Agile Practices and Tailoring of Practices 67

5 Discussion 83
5.1 Problem Statement 83
5.1.1 Characteristics of the Agile Method Tailoring Approach 83
5.1.2 Combination of Agile Practices 88
5.2 Implications of the Research 94
5.3 Research Limitations 97

6 Conclusion 99
6.1 Problem Statement 99
6.2 Future Work 103

Bibliography 105

A Appendices 111
A.1 Appendix A: Interview Guides 111
A.2 Appendix B: Agile Practices Survey 118
A.3 Appendix C: Confidentiality Agreement 120

List of Figures

2.1 The waterfall model (Royce, 1987) 12
2.2 Taxonomy of ISD agility (Conboy, 2009) 16
2.3 The Scrum development process framework (Scrum.org, 2017) 20
2.4 A typical Kanban board (Kniberg and Skarin, 2010) 26
2.5 A DevOps approach showing development and operations as a continuous and integrated process (Atlassian, 2018) 29
2.6 Degree of adaptability of agile software development methods (Kniberg and Skarin, 2010) 34
2.7 Contingency factors process for method selection (Campanelli and Parreiras, 2015) 36
2.8 Method engineering process for method definition (From Campanelli and Parreiras, 2015 based on Brinkkemper, 1996) 37

3.1 Node hierarchy used for coding the textual data 53

4.1 Overall project organization 61
4.2 Cumulative flow diagram from the project 72
4.3 Sample configurable Kanban board layout from the project 73
4.4 The over time development towards more self-organizing and cross-functional teams 75
4.5 Important functions of stand-up meetings in the project 77

List of Tables

2.1 Traditional vs agile software development (excerpt from Nerur, Mahapatra, et al., 2005) . . . 15

3.1 Databases used for literature search . . . 42
3.2 Characteristics of the research methodology (Runeson and Höst, 2008) . 45
3.3 Key figures related to interviews and resulting material . . . 50
3.4 Overview of the roles of informants and the number of informants and interviews for each role . . . 51

4.1 Overview of key stakeholders in the project . . . 58
4.2 Overall role distribution in the project . . . 60
4.3 Overview of identified agile practices from survey and additional practices (Survey in Appendix A.2, based on Rodríguez, Markkula, et al. (2012)) . 69

Introduction 1

1.1 Background and Motivation

Information technology have become increasingly important for organizations and individuals in society today. In association to this, software products and services have come to play a key role, as is evidenced in the high investments in software development projects in both the public and private sectors. However, the software industry is known to be exposed to high amounts of change, which among other factors have contributed to numerous challenges and also project failures (Highsmith and Cockburn, 2001; Jørgensen, 2016). Some failures have been related to the project methods used and their ineffectiveness in turbulent business environments. As a result, this have led to development of project methods for software development that are more capable of handling change and other known challenges. These are known as agile methods, and have proved to be beneficial for managing software development projects which often involve high risk and unstable requirements.

A collection of agile methods and practices have emerged that have become commonplace in practice. Examples of long-standing and widely used agile methods are Extreme Programming (XP) and Scrum (Abrahamsson, Salo, et al., 2002; Rodríguez, Markkula, et al., 2012; VersionOne, 2017). Founded on the principles of the agile manifesto (Fowler and Highsmith, 2001), agile methods and their underlying practices embrace change through frequent delivery of working software, active customer involvement, and team autonomy (Dybå and Dingsøyr, 2008; Lee and Xia, 2010). Studies suggest that practitioners experience many advantages using agile approaches, such as increased client benefits (Dybå and Dingsøyr, 2009; Jørgensen, 2016). The positive results have aroused great interest among many researchers and led to an established field of research. Despite this, the area is still under constant development and change, and new concepts and phenomena emerge that need to be explored. As pointed out by some proponents, the area is characterized as very practitioner-driven and research tend to lag behind as new themes emerge in practice. This leaves behind a backlog of research issues, and a focus on empirical research have been requested (Dingsøyr, Nerur, et al., 2012; Dybå and Dingsøyr, 2008).

The constant change in the area of software development is shown by a recent trend observed in both practice and research. Organizations and projects that develop software are steadily exposed to more demanding customers. Moreover, more business- and safety critical systems require errors to be quickly fixed and resolved. This presses forward a need to be able to create and handle change even faster than before in order to stay competitive in industries and markets. These are among factors that now motivate practitioners to adopt agile approaches that enable more rapid and frequent delivery of software features all the way from development to execution. Fitzgerald and Stol (2014) argue that this trend is manifest in developers' increased use of certain practices and techniques. Some good examples are DevOps, Continuous Integration and Continuous Deployment which focus on removal of barriers and streamlining of development processes for quicker end-to-end delivery. Dingsøyr and Lassenius (2016) found that there have been a sharp increase in interest for DevOps and related practices since 2012. Furthermore, agile methods such as Kanban (Kniberg and Skarin, 2010) is increasingly used as a complement or alternative to other agile methods in this context (Fitzgerald and Stol, 2014). According to the annual State of Agile Report by VersionOne, the use of Kanban techniques among respondents increased from 39% to 50% from 2015 to 2016, while 71% of respondents had or planned to introduce DevOps initiatives in their organizations (VersionOne, 2017). Overall, these figures and recent research indicate a movement towards adoption of agile approaches that enable even greater agility and other benefits in practice.

Dingsøyr and Lassenius (2016) describes the trend as a transition in agile software development from what has been initial recommendations of monthly iterations with Scrum, towards continuous delivery of software features. They argue that this idea is in fact old, but that the possibilities have increased with maturing technology. However, the state-of-the-art in agile software development is driven by the industry and remains to be more researched (Dingsøyr and Lassenius, 2016). Based on this and the above, one of the main motivations for this study is to investigate agile methods in the context of this trend where changes and new features need to be created, handled and delivered more rapidly and frequently than before.

Most agile methods such as Scrum and XP were originally designed with respect to a time-based iterative model where software features are delivered, or at least completed and shippable, every two to four weeks. The question is then whether new methods are required or if the existing ones may be used and tailored with the aim of continuous software delivery. This brings us to another stream of research which is not directly related to recent trends, but is central to the background and motivation of this study; namely research on *method tailoring*. When practitioners adopt methods and report promising results, others often reuse these with the hope that they will improve their own projects. However, there has been a long-standing acknowledgement that project methods are not universally applicable in their original format due to differences in context (Conboy and Fitzgerald, 2010). This is perhaps particularly true for software development projects because of the high complexity and changeability of software (Brooks, 1987). As a result, software development methods should be tailored to the actual needs of the contexts in which they are to be used to achieve optimum effect. This have come to represent a separate stream of research which have focused on how software methods, including agile methods, may be tailored (Campanelli and Parreiras, 2015). Much of this research focus on possible approaches to method tailoring that may be used in practice. Conboy and Fitzgerald (2010) illustrates one approach where an existing agile method is selected and tailored to better suit a project's needs. Another possible approach is to tailor a method through a combination of practices from a selection of agile methods (Fitzgerald, Hartnett, et al., 2006). Moreover, tailoring a method may not only make it a better initial fit, but can directly contribute to increased agility by actively tailoring the method to changes in the environment (Henderson-Sellers and Serour, 2005; Turk et al., 2002). Method tailoring is arguably therefore very relevant in today's challenging business environment where change is more present, and where there is more pressure on being agile in both organizations and projects.

Tailoring of agile methods is a research theme that have gradually gotten increased attention, but the total amount of research is still relatively small (Campanelli and Parreiras, 2015). Some of the existing research have focused on more traditional agile contexts with emphasis on how specific methods such as Scrum and XP may be tailored (e.g., Conboy and Fitzgerald, 2010; Fitzgerald, Hartnett, et al., 2006). Recent publishing have also studied tailoring of agile methods for use in large-scale projects (Rolland et al., 2016). This master thesis, however, will study method tailoring in the context of the emerging trend towards continuous software delivery in the software industry. Very little research was identified which study method tailoring in this modern and relevant context.

One last notable motivator for the chosen research direction was the case which is studied in this master thesis. I was lucky to come across a case that had adopted and tailored an agile method where software was delivered on a daily (i.e. continuous) basis. While agile method tailoring could have been studied in a more traditional agile project, it was arguably more interesting and relevant to study this in a more modern setting.

1.2 Problem Description

In the previous section (Section 1.1) I mentioned the request and need for more empirical research on topics related to agile software development. Then, a background and a motivation for research on agile method tailoring in the context of recent trends was given. In this study, this context will be viewed as a trend towards *continuous software delivery*. In order to clarify some of the terms and words used in the following problem statement and research questions, clarifications are made in Section 1.2.1. Some justifications for and explanations of the research questions are also provided after each question.

Based on the background and motivation, the following problem statement was chosen:

> *How can agile methods be tailored with aim towards continuous software delivery in projects?*

In order to illuminate and answer the problem statement, two research questions was formed to be investigated in a real case:

> **RQ1:** What characterizes an agile method tailoring approach in a modern software development context?

An essential part of tailoring an agile method in practice is the approach used. Method tailoring approaches have previously been a topic of research and review, and deals with how practitioners may tailor software methods to a situation (Campanelli and Parreiras, 2015). From my own literature review, it was evident that little research have been conducted on method tailoring and possible approaches in modern settings where software is delivered on a continuous basis. An *approach* is here not strictly defined, but what I am looking to investigate is the prominent characteristics of how agile method tailoring can be conducted in practice. Some existing tailoring approaches previously suggested in literature will be provided in the theory chapter (Section 2.3) and used to support the analysis and discuss findings.

> **RQ2:** How were practices combined into an agile method with aim towards continuous software delivery?

An additional contribution to answering the problem statement is an analysis of the resulting tailored method in the studied case. In this regard, I will focus on the agile practices that were prominent and which contributed to the ability to deliver software on a continuous basis. Little empirical research was identified that investigate how agile methods and practices can be combined (thus tailored) into an agile method in contexts where software is developed and delivered continuously.

1.2.1 Clarifications and Definitions

Two clarifications will be made regarding some of the terms and words used in the problem statement, research questions, and the thesis throughout. This should also contribute to a clearer understanding of how I intend to answer the problem statement and research questions.

Method tailoring

Existing literature have used several different terms interchangeably for the notion of tailoring methods to different situations. Some examples are *method tailoring*, *method adaptation*, *method configuration*, and *method customization* (Aydin et al., 2004; Conboy and Fitzgerald, 2010; Fitzgerald, Hartnett, et al., 2006). It seems to be that researchers have applied different terms for the same concept, with slight variations. However, the term *method tailoring* is prominently used as an umbrella term for the domain and its underlying concepts. Thereby, *method tailoring* is also used as an umbrella term in this study. For the purpose of this study, a definition of a *method* and *method tailoring* is provided. A *method* will be defined as:

> *"an approach to perform a systems development project, based on a specific way of thinking, consisting of directions and rules, structured in a systematic way in development activities with corresponding development products"* (Brinkkemper, 1996)

Furthermore, *method tailoring* will be defined as:

> *"a process or capability in which human agents through responsive changes in, and dynamic interplays between, contexts, intentions, and method fragments determine a systems development approach for a specific project situation"* (Aydin et al., 2004).

According to Aydin et al. (2004), *method fragments* can be a description of a software development method, or any coherent part thereof. It can be principles, fundamental concepts, products to be delivered, development activities, techniques, tools, etc. Method fragments is a central concept in method tailoring theory, which will be covered in Section (2.3.2).

Continuous software delivery
The second clarification is regarding the meaning and scope of the words "*continuous software delivery*" in the problem statement and research question.

The focus of this study is to investigate tailoring of agile methods, but more specifically in the context of the recent trend observed in practice. That is, the change in agile approaches from what have been typical recommendations of iterative deliveries every two to four weeks (frequently), towards delivery of software features up to multiple times a day (continuously).

An increasing number of software development approaches and principles tend to include the word "*continuous*" nowadays. Examples are Continuous Integration (CI), Continuous Deployment, and even Continuous Delivery (CD) (Fitzgerald and Stol, 2014). These are fairly technical engineering practices, but play an important role in improving the speed of software delivery in modern projects. However, it should be noted that the problem statement and research question does *not* refer specifically to any of these practices. In particular, the practice of Continuous Delivery (CD) should not be confused with the words "continuous software delivery" used in this study. That is not to say that none of these practices will be brought up, as they are important agile practices in this context. That said, this study will mainly focus on agile methods and tailoring from a management perspective, and not dive into many details on technical engineering practices such as those above.

1.3 Scope

Some constraints exist in the scope of this study. First and foremost, the main purpose have been to answer the problem statement by more specifically answering the underlying research questions. When doing so, this mainly involves concepts related to agile methods and method tailoring. In general, software engineering can be regarded as a wide and multi-disciplinary field, with management being one important aspect. This study will be done from a management perspective, and will not dive into much detail on aspects such as technical development practices, techniques, or technologies. Concepts related to areas such as organization and contracts have also been raised in this study when relevant, but without going into particular detail. The concepts that are most important for this study are introduced in the background literature and theory in Chapter 2.

The problem statement and underlying research questions will be answered based on the findings that were considered relevant from the analysis of collected data, and the available theoretical basis provided in Chapter 2. Much data were collected about the studied case and analyzed, but only the findings relevant within the scope of the research have been included in the results in Chapter 4.

The purpose of this study was not to develop new theories, critically test existing theories, nor attempt to make many bold generalizations beyond the studied case. Instead, it aimed at exploring a modern development context to contribute to research and practice with new and hopefully valuable, empirical insight. Towards the end of the discussion chapter, an evaluation of the implications of this study towards practice and research is done (Section 5.2).

1.4 Target Audience

This study is targeted at multiple audiences. First and foremost, it should be of great interest to agile communities, which also includes the project management community. This includes both researchers and practitioners who have an interest in agile software development and method tailoring.

Researchers may have particular interest in this work if they seek a thorough description of a real world case or inspiration for future research directions on method tailoring and agile methods and practices. Empirical research such as this can be of high value, and only a limited amount is currently available on these topics.

Practitioners that are either novice or experienced in agile software development, but who seek new information and inspiration for their software development projects should also find this work relevant. It may be particularly relevant for project managers whose role is important when it comes to facilitating the use and tailoring of project methods, such as agile methods, in development projects.

It should be made clear that the reader is expected to have some understanding of fundamental ideas in agile software development, agile methods, and project management. In addition, the reader should possess some knowledge on related concepts such as information technology.

1.5 Thesis Structure

Below is an overview of the remaining chapters of this thesis and what the reader can expect to read about in each of the subsequent chapters.

Chapter 2 - Background Literature and Theory
Includes important aspects related to agile software development, an overview and description of relevant agile methods and practices, and an introduction to method tailoring and known tailoring approaches. The purpose is to provide the reader with a good understanding of these concepts, and also serve as a theoretical basis for discussion of results.

Chapter 3 - Method
Contains detailed descriptions, explanations, and justifications on how the study was conducted. The whole process from conducting the literature review, data collection, analysis and reporting is covered. The last two sections deals with how the research complied to confidentiality agreements, and tactics that were used to improve research validity.

Chapter 4 - Results
Presents the results from the analysis of the collected data. An overview of the studied project is first presented to provide initial context. Then the results directly related to method tailoring and the agile method and practices in the case are presented.

Chapter 5 - Discussion
Contains the discussion of the results towards each of the two research questions and theory. The second last section discusses the implications of the research to research and practice. The last section provides an assessment of the limitations of the research.

Chapter 6 - Conclusion
Presents the conclusions of the study with regard to the problem statement and the

research questions. Each of the two research questions are answered based on what has been learned throughout the study, with emphasis on the discussion. Lastly, suggestions for future work are presented.

2 Background Literature and Theory

The purpose of this chapter is twofold. For one, provide the reader with a relevant background on key concepts of the study through a presentation of existing literature and theory. Secondly, it will serve as a theoretical foundation for discussing the results later in Chapter 5.

The chapter is comprised of four main sections. The first section introduces agile software development and important aspects and issues. In the second section, agile methods and practices are presented with focus on those that are most relevant for this study. The third section is dedicated for introducing method tailoring and scientific references related to this. Lastly, a short summary is done with the intention to recap important concepts of the chapter and clarify how these fits in with the rest of the research.

2.1 Agile Software Development

In this initial section an introduction to agile software development will be presented. This includes a brief comparison to its predecessor, traditional software development, and some benefits and challenges related to agile software development. After this, agile principles and a definition of agility will be provided. Lastly, a brief background on the emerging trend towards continuous software delivery is included.

2.1.1 Traditional versus Agile Software Development

Software development is generally conducted under what can be said to be two paradigms: traditional software development and agile software development. Both are used in practice today. However, agile software development arose as a response to the challenges of traditional development, and is considered to be an improvement in many contexts. The foundation for agile software development was laid in 2001 with what is known as the agile manifesto (Fowler and Highsmith, 2001). Prior to this movement, the alternatives to software development were traditional approaches.

Several approaches to software development fall within this category, with the Waterfall model being one much used example (Royce, 1987). Common to these are that they are characterized as plan-driven and that they place much emphasis on extensive documentation. The essential purpose of this is to avoid changes to occur, thus reducing costs (Highsmith and Cockburn, 2001).

In the waterfall model, the requirement specification is formulated in detail in co-operation with the client before development has begun. Then the product follows a set of predefined and sequential phases in a manner that resembles a *waterfall*, before a complete product is completed and delivered (Royce, 1987). This has been criticized because the model does not involve the customer other than in the initial stages of the development process (Petersen et al., 2009). As software development projects become larger and more complex, it becomes difficult to define all requirements in advance. In addition, there is an increased risk that major changes must be made in later stages, which can be costly and end with unsatisfactory results. There are also many dependencies between the sequential phases, which means that the development model provides little flexibility.

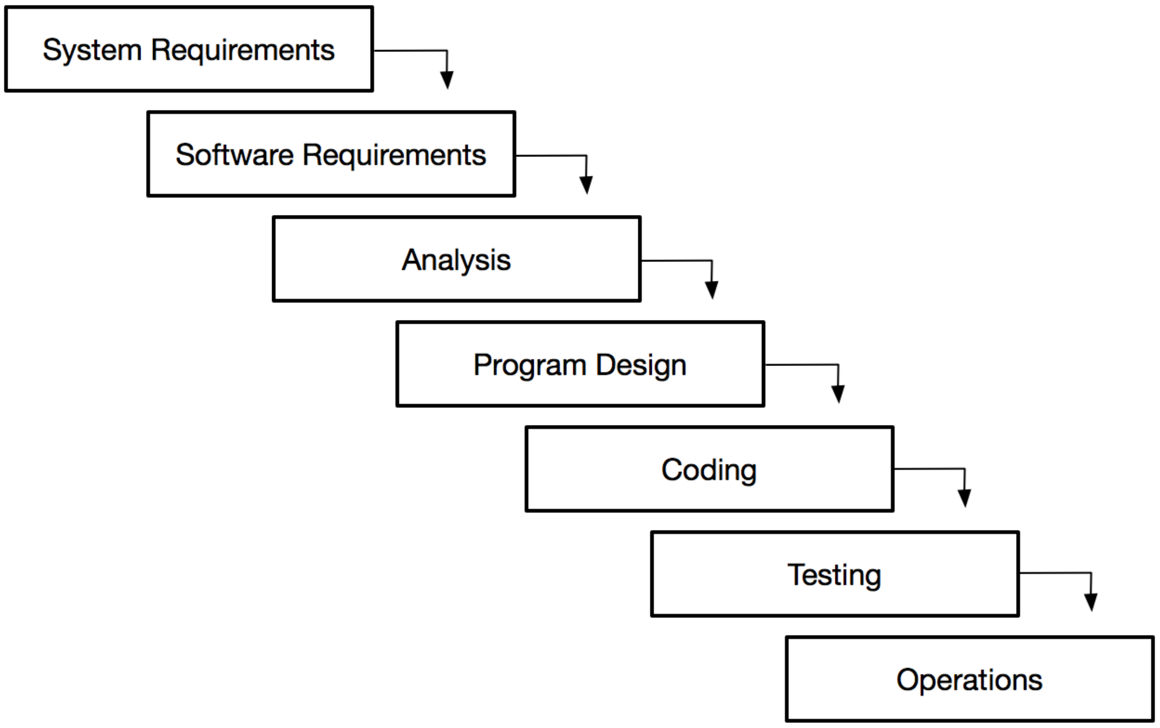


Figure 2.1.: The waterfall model (Royce, 1987)

Projects became more complex and demanding in the 1990s, and developers experimented with new approaches to software development. This eventually led to the movement known as agile software development. Agile software development improved on traditional approaches by recognizing that project requirements can not all be foreseen prior to development. In addition, agile software development emphasizes delivery of high quality software through creativity and collaboration among developers, and cross-functional and self-organizing teams that possess the necessary skills and empowerment to manage and handle their own work (Highsmith and Cockburn, 2001; Lee and Xia, 2010; Moe et al., 2009). Instead of planning for changes that may occur, the goal is to handle changes as they arrive in a cost-effective manner (Williams and Cockburn, 2003). Agile software development is primarily based on a set of principles and values. Several agile methods have emerged as a result of the agile software development paradigm. However, many of these are based on previous methods and practices. The difference with agile methods, is that they adhere to agile principles and values (Cohen et al., 2004). These are summarized in the four core values in the agile manifesto (Fowler and Highsmith, 2001):

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

While the items to the left are most valued, agile software development also value the items to the right. Thus, agile software development does not abandon activities such as planning and documentation, but keeps it to a minimum (Fowler and Highsmith, 2001).

Research indicate that agile software development have been met with much positivism. Practitioners are more satisfied, achieve better results, and deliver increased client benefits using agile approaches (Dybå and Dingsøyr, 2009; Jørgensen, 2016; Rodríguez, Markkula, et al., 2012). Some reported benefits have appeared in easy adoption, customer collaboration, work processes for handling defects, learning among developers, thinking ahead of management, focusing on current work for engineers, and software estimation (Dybå and Dingsøyr, 2009). Other prominent positive effects are improved team communication, enhanced ability to adapt to changes, and increased productivity (Rodríguez, Markkula, et al., 2012). However, there are also numerous issues and challenges that are being studied in relation to the area (Dingsøyr, Nerur, et al., 2012). A key element in agile software development is more frequent delivery of working software and increased customer focus through frequent involvement of customers and stakeholders. Adequate customer collaboration have consequently been considered a key success factor in agile software development (Hoda et al., 2011; Misra et al., 2009). Jørgensen (2016) found that agile projects where the customer is involved through frequent deliveries, is open to a flexible scope, and avoid fixed-price contract types increases the likelihood of project success. Vinekar et al. (2006) even argue that agile methods are dependent on an on-site customer to identify and prioritize features, provide feedback, and guide change throughout the course of development. With demands such as these, organizational culture and other external factors can be a challenge. This can impact the deployment of agile methods and practices in various ways, and causes challenges that requires organizations to adapt to agile ways of working (Gandomani et al., 2013; Gregory et al., 2016; Nerur, Mahapatra, et al., 2005). Agile software development also requires forms of management and control other than what have been previously seen in traditional approaches. Instead of a command-and-control control regime like in traditional approaches, agile software development prefer leadership and collaboration (Nerur, Mahapatra, et al., 2005). Other known barriers to agile adoption can also be found within categories such as process, people and technology (Gandomani et al., 2013; Nerur, Mahapatra, et al., 2005). Overall, this shows that agile software development are not a phenomena without flaws despite the many benefits over traditional approaches.

A comparison of characteristics between traditional and agile software development is summarized in Table 2.1.

2.1.2 Agile Principles and Agility

The agile manifesto further contain 12 principles. These have since served as a basis for the many agile methods and practices that practitioners adopt. However, the principles are not a formal definition of *agility*, but represent guidelines for developers who want to practice agile development (Dingsøyr, Nerur, et al., 2012). The principles are written as follows (Fowler and Highsmith, 2001):

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

	Traditional	**Agile**
Fundamental Assumptions	Systems are fully specifiable, predictable, and can be built through meticulous and extensive planning.	High-quality, adaptive software can be developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change.
Control	Process centric	People centric
Management Style	Command-and-control	Leadership-and-collaboration
Role Assignment	Individual — favors specialization	Self-organizing teams — encourages role interchangeability
Communication	Formal	Informal
Customer's Role	Important	Critical
Project cycle	Guided by tasks or activities	Guided by product features
Desired Organizational Form/Structure	Mechanistic (bureaucratic with high formalization)	Organic (flexible and participative encouraging cooperative social action)

Table 2.1.: Traditional vs agile software development (excerpt from Nerur, Mahapatra, et al., 2005)

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity–the art of maximizing the amount of work not done–is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Many agile methods have emerged to include practices based on these principles such as frequent software delivery, active customer participation, self-organizing and cross-functional teams, and continuous improvement with retrospectives. Some of these are further examined in Section 2.2.

Agility has been a key concept in other areas of business and management prior to agile software development (Cockburn, 2002; Conboy, 2009; Nerur and Balijepally, 2007). There is still a debate in the research community about what *agility* in the context of software development actually entails. As a result, several definitions have been proposed (Dingsøyr, Nerur, et al., 2012). Conboy (2009) proposed a definition and a taxonomy of agility in context of information systems development (ISD). Based on the related concepts *flexibility* and *leanness* from the fields management and manufacturing respectively, he developed a definition for agility as (Conboy, 2009, p. 336):

> "*the continual readiness of an ISD method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment.*"

Based on the definition, he also developed a taxonomy for ISD agility, which was also tested on two software development projects in practice. The taxonomy is presented in Figure 2.2. In the taxonomy it is emphasized that agility, as defined, is achieved

1. To be agile, and ISD method component* *must* contribute to one or more of the following:
   a) creation of change
   b) proaction in advance of change
   c) reaction to change
   d) learning from change
2. To be agile, an ISD method component *must* contribute to one or more of the following, but *must not* detract from any:
   a) perceived economy
   b) perceived quality
   c) perceived simplicity
3. To be agile, an ISD method component *must* be continually ready i.e. minimal time and cost to prepare the component for use.

* An ISD method component refers to any distinct part of an ISD method.

Figure 2.2.: Taxonomy of ISD agility (Conboy, 2009)

through the collection of components in a software development method. *Method* *component* refers to the different parts that a method may consist of, such as practices. To elaborate on the taxonomy, a method component must contribute to agility through creation of, handling, or learning from change. It must also contribute to both economic benefits, quality, and simplicity. For example, if extensive documentation contributes to quality and simplicity, but is not perceived to have economic benefits due to rapid obsolescence, the practice is not regarded as agile (Conboy, 2009). Finally, it is stated that agility is achieved if a method component is practical and efficient in use. Conboy, 2009, p. 341 exemplifies this: "*Continual readiness of the method component is also a prerequisite. For example, acceptance tests certainly contribute to agility in some circumstances; but if it takes hours to prepare tests every time they run, then their contribution to agility is unclear*".

Conboy (2009) states that some methods and practices that appear as agile are not necessarily so in any given project. This suggests that tailoring of methods is necessary to maintain agility. This is also supported by Turk et al. (2002), who argue that the degree of agility in a development process can be defined based on a project team's ability to dynamically adapt the process based on changes in the environment. They imply that high levels of agility depend largely on adaptive methods and experienced developers who know how to tailor the methods effectively. Similarly, Henderson-Sellers and Serour (2005) argue that agility involves the ability to adjust and fine-tune development methods as needed, as well as handle requirement changes. In other words, researchers claim that agility in practice involve more than strictly following a set of predefined agile methods, but requires active supervision and tailoring of these to be agile.

2.1.3 Towards Continuous Software Delivery in Agile Software Development

The meaning and practice of agile methods has been constantly changing. Baskerville et al. (2011) found that the development of agile approaches seemingly followed a repeating two-stage pattern between *change in context* and *change in process*. The reason seemed to be that changes in market situations and circumstances affect how agile approaches are practiced and evolved. This is interesting, because it brings us to newer trends which shows that agile software development experiences a shift in the meaning of "frequent delivery". For long "frequent delivery" have been used in agile software development to refer to software delivery typically every couple of weeks to once every month. This is evident in the agile manifesto (Fowler and Highsmith, 2001), as well as long-standing agile methods such as Scrum (Schwaber,

2004) and Extreme Programming (XP) (Beck, 2004). For many years, this delivery rate have evidently provided big improvements over traditional methods such as the waterfall model (Royce, 1987). However, newer trends show a replacement of the word "frequent" with the word "continuous" to refer to a change towards delivery on a daily basis (i.e. continuously). This trend has been discussed by researchers such as Fitzgerald and Stol (2014), who see this trend with a holistic perspective. They also refer to a family of practices, that have become central in this shift towards even more rapid delivery. Among these are the fairly technical engineering practices Continuous Integration and Continuous Deployment, which are also important in the emerging DevOps phenomenon (Section 2.2.3). They also mention Kanban (Section 2.2.3) as an agile method that have been increasingly used in conjunction with these practices. These terms will all be described in greater detail later in this chapter. Overall, this trend suggest that there is a change in context where there is a push towards being able to deliver with a shorter time-to-market and more frequent than before. Ultimately, this makes it possible for organizations to more quickly handle change and stay more competitive, but also depend more on agile methods to provide sufficient agility in project contexts.

2.2 Agile Methods and Practices

This section will first present an overview of agile methods and a general introduction to what we know about these. After this, some of the most used agile methods and underlying practices will be described in more detail with emphasis on the ones that are most relevant to this study. These have been categorized under two overarching classifications; *iterative methods* and *flow-based methods* respectively. Lastly, a section is dedicated to present an overview of important agile practices for this study.

2.2.1 Agile Methods Overview

Since the origin of the agile manifesto, a variety of different agile methods have emerged to guide development of high-quality software in an agile manner. Some widely used agile methods are Scrum, Extreme Programming (XP), Agile Modeling, Feature Driven Development (FDD), Kanban, Adaptive Software Development, Dynamic Systems Development Method (DSDM), and Test Driven Development (TDD) (Rodríguez, Markkula, et al., 2012).

Many agile methods are typically based on an incremental and iterative model for software development. This means that software development is broken down into increments where batches of software features are designed, developed, tested, and sometimes delivered, in repeated cycles. Recommended iteration lengths vary from one week to six weeks depending on the method (Abrahamsson, Salo, et al., 2002). This can enable adequate stability and predictability in the development environment, while at the same time support frequent releases to get feedback and adjust to change. Among the above list of widely used methods most of them are based on an iterative model, with Scrum and Extreme Programming (XP) being the two that are seemingly most used in practice (Rodríguez, Markkula, et al., 2012; VersionOne, 2017).

Furthermore, methods have started to appear that are based on another model, namely a flow-based development model. Unlike an iterative model, a flow-based model establishes design, development, testing, and delivery of independent software features as a continuous process. In other words, software features are not batched together as a larger increment to be completed within a given iteration length (Fitzgerald and Stol, 2014). In practice, this can be understood as each individual feature being one very small increment which has its own very short iteration. This can have advantages over the iterative development model described above, such as more flexibility in the development process (Birkeland, 2010). Among the above list of widely used methods, Kanban is the only method that may be classified as flow-based. The concept of "flow" will be further described in the section on Kanban (Section 2.2.3).

Overall, there are differences in all of the agile methods. Only a couple of methods, such as Dynamic Systems Development Method (DSDM), cover all development phases fully. Other methods only focus on some aspects of development. For example, Scrum and Kanban mainly covers aspects related to project management, while Extreme Programming (XP) focus more on fairly technical software development practices (Abrahamsson, Salo, et al., 2002; Kniberg and Skarin, 2010). The fact that many methods only cover certain aspects means that one method alone may not be able to cover all of a projects needs. This is one of the main motivations for method tailoring, which will be covered in Section 2.3.

2.2.2 Iterative Methods

This section presents the two agile methods Scrum and Extreme programming (XP), which are most relevant for this study. Common to these two are that they adopt an iterative model to software development.

Scrum

Scrum is a software development process framework for incrementally and iteratively building software in complex environments for small teams (Rising and Janoff, 2000; Schwaber, 2004; Sutherland, J. and Schwaber, K., 2016). The framework is comprised of a set of roles, artifacts, events, and guidelines for software development. According to research and surveys it is the most adopted agile method by practitioners today by a seemingly large margin (Rodríguez, Markkula, et al., 2012; VersionOne, 2017).

Collectively, the roles in Scrum are all part of The Scrum Team. This include the Product Owner, Scrum Master, and The Development Team. The artifacts are the Product Backlog, Sprint Backlog, and the Increment. And the events are the Sprint, Sprint Planning, Daily Stand-ups, Sprint Review, and Sprint Retrospective. The overall process and how these components relate is illustrated in Figure 2.3.

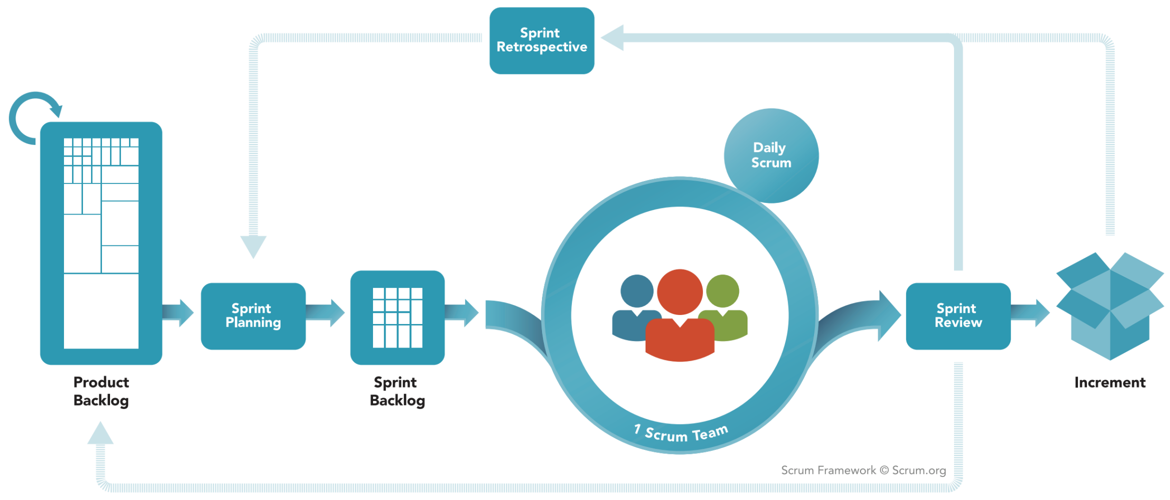


Figure 2.3.: The Scrum development process framework (Scrum.org, 2017)

The Scrum Process

In Scrum, increments of software is completed in iterations called *Sprints*. The goal of a Sprint is to finish developing a predecided set of software features within a time frame of two to four weeks. Product features are kept in the Product Backlog, which contains all the known and remaining work to be done in a project. The features are prioritized and managed by a Product Owner. Each Sprint, a collection of high-priority features are selected from the Product Backlog by the Scrum Team to be implemented. These are kept in a Sprint Backlog, which represents the work to be done in a given Sprint. This is done in the Sprint Planning meeting (Schwaber, 2004; Sutherland, J. and Schwaber, K., 2016).

The software is developed by the Development Team which is self-organizing and cross-functional. Self-organizing teams are empowered by the surrounding organization to make their own decisions on how to complete the work within a Sprint. Cross-functional teams consist of the necessary competencies to accomplish their work without external support. The only arena for day-to-day coordination between team members is the Daily Scrum, often called Daily Stand-up. Scrum emphasize that more time should be spent on creating working software, and less time in meetings. Daily Stand-up have a recommended duration of 15-minutes. As is implied, the event is performed while standing. The idea is that this leads to discomfort in the long run, thus minimizing the duration of the event. Each developer addresses the following three questions during the Daily Stand-ups:

1. What did I do yesterday that helped the Development Team meet the sprint goal?
2. What will I do today to help the Development Team meet the sprint goal?
3. Do I see any impediment that prevents me or the Development Team from meeting the sprint goal?

In larger projects with multiple Scrum Teams, a Scrum of Scrums meetings may be used for inter-team coordination in addition to Daily Stand-ups within each team (Schwaber, 2004). One way to do this is to have selected representatives from each team attend this meeting a couple of times a week, which have previously been found to be effective for inter-team coordination in practice (Dingsøyr, Moe, et al., 2017).

When a Sprint period have reached its end, a Sprint Review is performed where the team, customers and other key stakeholders are involved for a more comprehensive demonstration and inspection of the work. This is an opportunity for collecting feedback, which can be used to adjust the Product Backlog according to customer needs. This can make Scrum suitable for situations where it is difficult to plan ahead, since planning and scope adjustment can be done frequently throughout the project.

The Sprint Retrospective is the last event of a Sprint, which is a session where the team inspects itself and reflects on possible improvements, both in the process and in the backlog. Thus, frequent attention to improvement is a key part of Scrum.

In Scrum, the Scrum Master represents a dedicated role for facilitating the Scrum Team throughout the development process. The responsibility is mainly aiding the team members to practice Scrum correctly, and remove impediments in the process.

Importantly, the Scrum Master is not a project manager, and should not control nor manage the team's work (Schwaber, 2004; Sutherland, J. and Schwaber, K., 2016).

Extreme Programming

Extreme Programming (XP) is an iterative agile method that focuses on best practices for software development. It was first introduced in 1999 (Beck, 1999), and is still one of the most widely used agile methods (Rodríguez, Markkula, et al., 2012; VersionOne, 2017). While Scrum is more project management oriented and exists as a process framework, XP is more concerned about the use of different practices to facilitate frequent and cost-efficient delivery of working software. There are 12 different practices in XP. Although it has been revised before to include more practices (Beck, 2004), this section will concentrate on the original 12 practices which are most widely used (Conboy and Fitzgerald, 2010). In addition to being known for focusing on concrete practices, XP is also described as a philosophy and set of principles. Beck (2004) describe XP to include:

- A philosophy of software development based on the values of communication, feedback, simplicity, courage, and respect.
- A body of practices proven useful in improving software development. The practices complement each other, amplifying their effects. They are chosen as expressions of the values.
- A set of complementary principles, intellectual techniques for translating the values into practice, useful when there is not a practice handy for your particular problem.
- A community that shares these values and many of the same practices.

Extreme Programming also emphasize that its practices are meant to be picked and used based on the context at hand (Beck, 2004). Thus, XP can be a well suited candidate for tailoring project specific agile methods (Abrahamsson, Warsta, et al., 2003; Fitzgerald, Hartnett, et al., 2006). XP emphasize simplicity and flexibility as described by Lindstrom and Jeffries (2004, p. 43): "Whereas many popular methodologies try to answer the question *what are all of the practices I might ever need on a software project?*; XP simply asks, *what is the simplest set of practices I could possibly need and what do I need to do to limit my needs for those practices?*". However, some have also pointed out that the method's practices are very much overlapping and complementary, and that they are most effective when used together. Thus, some argue that the benefits of XP is only achieved when all of it's practices are adopted as a whole (Fitzgerald, Hartnett, et al., 2006).

The 12 practices of XP are portrayed in the following list (based on Beck, 2004; Conboy and Fitzgerald, 2010; Dybå and Dingsøyr, 2008; Lindstrom and Jeffries, 2004):

1. **The Planning Game:** An activity for predicting what will be accomplished by the due date of an iteration, and determining what to do next. This includes *release planning* and *iteration planning*.
2. **Small Releases:** XP teams should break down tasks in order to be able to deliver small releases frequently. Typically every two weeks. This increases visibility of the work, and possibility for frequent feedback.
3. **Metaphor:** The development team develop a common understanding of the software and how it is supposed to work.
4. **Pair Programming:** Developers program in pairs on the same computer. This enables developers to communicate and review code, with the aim of producing better quality code.
5. **Simple Design:** XP teams develop software to a simple design. The developers should start simple, and improve the software through testing and design.
6. **Testing:** Developers continuously write tests as they make changes or create new functionality. The software is required to pass the tests to demonstrate working code before integrating the code, and that they meet customer expectations and specifications.
7. **Refactoring:** Code is frequently revisited, restructured, and improved to create better quality software. This can be particularly useful to improve non-functional attributes of software such as simplicity, flexibility, etc.
8. **Continuous Integration:** Code is integrated and built up to several times a day. By integrating small changes frequently into a common code base, developers can avoid serious problems or difficulties trying to integrate lots of accumulated work.
9. **Collective Code Ownership:** The developers in the team are all responsible for all the code. In practice, any developer can make changes anywhere in the system, any time.
10. **Sustainable pace:** The team works hard, but at a sustainable pace to avoid burnout in the long term.

11. **On-site customer:** All the contributors to an XP project sit together as members of one team - the Whole Team. This includes key customer representatives, who provides the requirements, sets the priorities, and steers the project.
12. **Coding Standard** Developers adhere to a common set of rules for producing high quality code.

2.2.3 Flow-based Methods

This section will present the flow-based methods that are relevant for this study. This includes Kanban and DevOps respectively. Already here, I want to point out that DevOps is fairly new and not clearly defined in research or practice as of yet. As a result it is neither properly established as a method at this point. However, some propose it as a method and it consists of practices and principles for software development on par with other methods. It is also related to a flow-based way of thinking.

Kanban

Kanban is an agile method that is flow-based and increasingly used in practice. It is still not as widely used as iterative methods such as Scrum and XP (Rodríguez, Markkula, et al., 2012; VersionOne, 2017).

Kanban means "signboard" in Japanese, and originates from Lean manufacturing. In Lean manufacturing, Kanban is a flow-control mechanism for a pull-driven production system. In upstream processing like manufacturing, a pull-system is one which activities are triggered by downstream processing signals (Ikonen et al., 2010), i.e. work moves upstream when there is capacity and need. The core concept in Kanban in manufacturing is to visualize and control the flow of work. Work is visualized using a board containing cards that represent the work, hence the name.

The ideas of Kanban have since found its way to agile software development. Poppendieck and Poppendieck (2003) suggested possible translations of Lean principles into agile practices for software development, which they coined as Lean Software Development. Fundamental to Lean, and "Lean Thinking", is the principle of *reducing waste*. That is, everything that does not add value to the delivered product should be avoided. There are also other Lean principles. Some examples are the principle to *deliver as fast as possible* for very frequent feedback, to *empower teams* to bring decision-making down to the team level, and *see the whole* to avoid optimization of only parts of the product at the expense of the whole. Kanban emerged as an agile approach for developing software according to some of these Lean principles (Ahmad et al., 2013; Kniberg and Skarin, 2010; Poppendieck and Poppendieck, 2003).

Another key concept in "Lean Thinking" is that of *flow* (Poppendieck and Poppendieck, 2003), which we have already touched upon. Fitzgerald and Stol (2014) described "flow" in development as all the connected set of value-adding actions performed on a software feature from design to deploy. They distinguished this from iterative methods by adding that *flow* implies that each task should essentially flow through the process one-by-one, and not be batched and queued together as larger deliveries. This also require the process to be established as an end-to-end concept which includes every activity from identification of a feature to deployment to a production environment (Fitzgerald and Stol, 2014). Kanban in software development can be used to visualize and measure the flow of work in such an end-to-end process.

Implementations of Kanban in software development is relatively simplistic compared to agile methods such as Scrum and XP (Section 2.2.2). Kanban mainly consist of three practices (Kniberg and Skarin, 2010):

1. Visualize the workflow
2. Limit work-in-progress (WIP)
3. Measure the lead time (i.e. average time to complete one task)

Work in Kanban are the tasks, e.g. software features, to be implemented in a project. Kanban suggests that work is broken down into smaller work packages. This makes it easier to limit the amount of work in progress, which again decreases lead time (i.e. makes delivery faster) and improves quality of work (Ahmad et al., 2013). Work is visualized in Kanban using a Kanban board. A Kanban board consists of a number of columns in which work packages flows through. Work packages are often represented as small descriptions in the form of cards in the columns. The idea is that cards, that is the work to be done, flow independently across the columns from start (e.g. backlog) to finish (e.g. deployment). Moreover, research have suggested that there is no agreed upon standard on how a Kanban board should be configured in software development. Some seemingly typical column categories used in Kanban boards in practice are "*specification/analysis*", "*build/development*", "*test/acceptance*", and "*deploy/release*" (Corona and Pani, 2012). One example of a Kanban board is illustrated in Figure 2.4.

Limiting work-in-progress (WIP) implies limiting the amount of cards that are present in each of the columns in the Kanban board at any one time. The idea is that too much active work-in-progress decreases transparency and focus. This may eventually lead to bottlenecks and waste, which again reduces productivity and flow (Ahmad et al., 2013; Kniberg and Skarin, 2010).

For measuring lead time, different tools are used in practice. One possibility is to use Cumulative Flow Diagrams (CFD), which may be used to visualize and measure WIP and average lead times. CFDs can be used to keep track of the accumulation of work in the different columns of a Kanban board. Thus, it is a practical tool for highlighting whether there are bottlenecks or other issues in the process (Corona and Pani, 2012).

Kanban is simplistic in that it only focuses on a few rules for visualizing workflow. Furthermore, it does not define specific roles or arenas for coordination, such as a Product Owner or Daily Stand-ups found in Scrum (Section 2.2.2). Neither does it suggest any best practices for developing software such as those found in Extreme Programming (Section 2.2.2). Its simplicity makes it relatively easy to implement and adapt (Kniberg and Skarin, 2010). It is also sometimes used in combination with other methods such as Scrum, sometimes separated out as its own method named *Scrumban* (Ahmad et al., 2013). In recent times, Kanban have been increasingly used in contexts that pursue continuous software delivery (Fitzgerald and Stol, 2014).

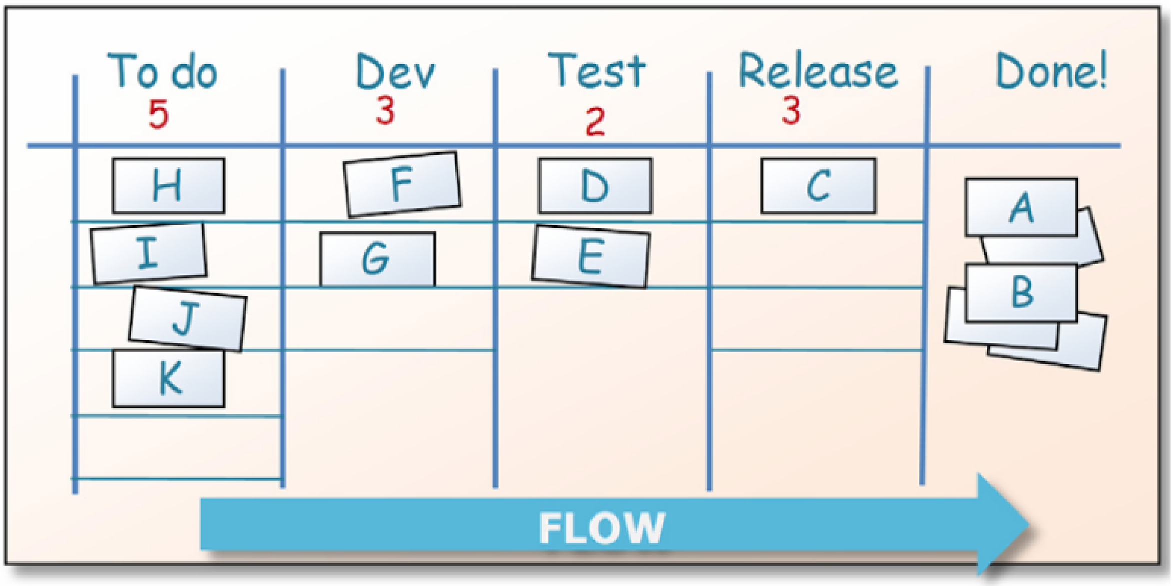


Figure 2.4.: A typical Kanban board (Kniberg and Skarin, 2010)

DevOps

DevOps is a term that has been popularized in the software industry in recent times, and is seeing increased use in agile organizations and projects (Brown et al., 2017; VersionOne, 2017). In practice, DevOps has evolved into involving several practices and principles (Lwakatare et al., 2016a). However, research also suggest it is still unclear how DevOps should be defined, and there seems to exist some blur regarding what it actually entails (Lwakatare et al., 2016b). A recent study proposed it as a promising early-stage project method, but acknowledged that this is not yet well established (Banica et al., 2017). Some other words used in literature to describe what DevOps entails include a culture, mind-set, principle, and a set of practices. According to Dingsøyr and Lassenius (2016), DevOps and related concepts is currently very practitioner-driven and research is lagging behind. All in all, DevOps seems to be a multi-faceted concept in an early stage of research. I will further present what seems to be the key elements of DevOps based on promising recent literature that has studied its application in practice.

DevOps is a merger of the words "developer" and "operations". In this case, developers are those who create software features, while operations personnel are those who ready, deploy and operate the software features and underlying systems. Traditionally, organizations have made a distinction between these into separate silos. However, it has been acknowledged that this represents a harmful disconnect between important activities (Fitzgerald and Stol, 2014). According to Humble and Molesky (2011), one main purpose of DevOps is to align the incentives of both developers and operations personnel. In practice, DevOps therefore promotes closer collaboration between or a merging of development and operation to remove what often represents a bottleneck between the two. This can make development and operations into an integrated end-to-end process. DevOps has an important place in the recent trend of continuous software delivery for this reason (Fitzgerald and Stol, 2014).

Some definitions of DevOps have been suggested by researchers. One suggested definition is "*a mind-set substantiated by a set of practices to encourage cross-functional collaboration between teams — especially development and IT operations — within a software development organization, in order to operate resilient systems and accelerate delivery of change*" (Lwakatare et al., 2016a, p. 7). Furthermore Lwakatare et al. (2016b) argue that the set of practices in DevOps originated from existing agile practices, and that it is also to a large extent informed by lean principles. According to Lwakatare et al. (2016a), the practices involved in DevOps varies in projects studied in practice. Through a study of DevOps among practitioners, they found that DevOps practices can typically be categorized into five main dimensions:

1. **Collaboration:** DevOps emphasize close collaboration between developers and operations personnel. This tend to require a rethinking and reorientation of roles and teams in development and operations activities in organizations and projects. Moreover, this can empower teams, especially developers, to gain more control over system operability which in turn may broaden their skillset and knowledge.
2. **Automation:** One goal of DevOps is to increase the speed and frequency of the delivery and deployment process. In order to achieve this, DevOps try to remove error-prone manual processes through automation practices. This emphasizes automation of build, test, and deployment processes. Typical agile practices that may be used for this are Continuous Integration which is found also in XP (Section 2.2.2) and the extended practice Continuous Deployment which is explained in the section on agile practices (Section 2.2.4).
3. **Culture:** Many argue that important to DevOps is also to have a focus on culture. It requires organizational and cultural change in order to create an empathic, supportive, and good working environment between development and operations. Empathy in a DevOps culture allows software developers and operators to help each other deliver the best software possible.
4. **Monitoring:** DevOps may support feedback loops through monitoring practices. This typically implies monitoring systems in order to detect potential defects early. This can be done through well-instrumented software systems and aggregate monitored data into rich insights. This can also potentially reduce the need for extensive testing before deployment.
5. **Measurement:** DevOps practitioners use metrics practices to monitor and assess the performance of their systems. Metrics may also be useful for assessing system quality and stability.

The proposed definition of DevOps and the five dimensions show principles and practices which have much in common with agile values and other agile methods. The integration between development and operation activities through DevOps can be viewed as a continuous process. Figure 2.5 illustrates a typical approach to development and operations in a DevOps environment.

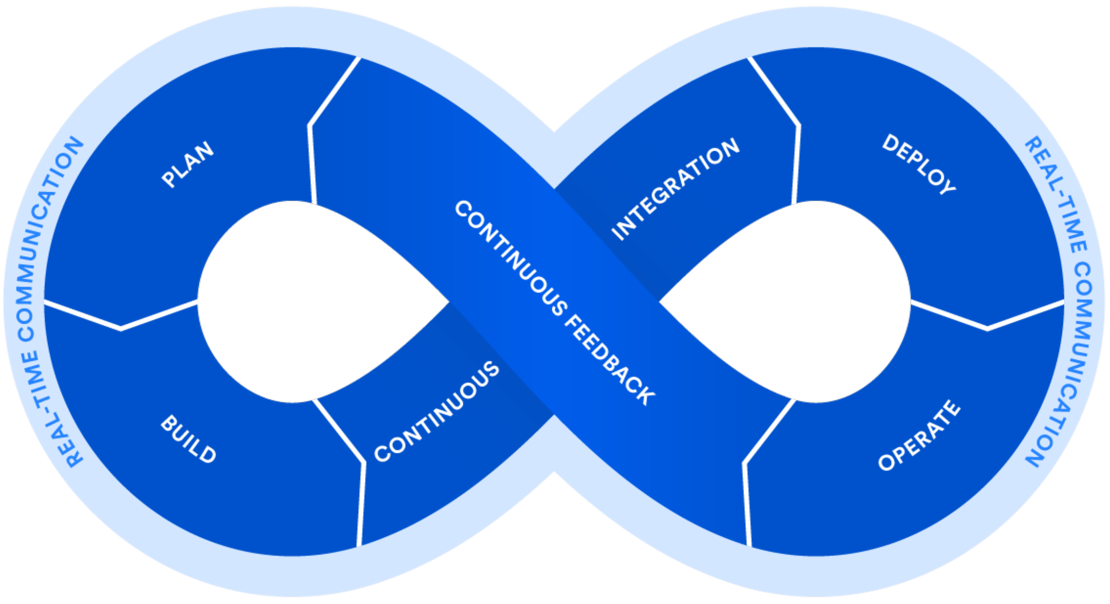


Figure 2.5.: A DevOps approach showing development and operations as a continuous and integrated process (Atlassian, 2018)

2.2.4 Agile Practices

Agile methods are essentially composed by agile practices. As previously mentioned, agile practices should help accomplish agility in a method (Conboy, 2009). Agile practices can be grouped into the three groups *management practices*, *software process practices*, and *software development practices* (Lee and Yong, 2013). Examples of agile management practices are on-site customer, daily stand-up meetings, self-organizing teams, and open work area. Software process practices can be to use simple design, collective code ownership, and frequent delivery of working software. Software development practices may be pair programming, unit testing and continuous integration to name a few. Collectively, these are among others that may be combined to guide agile software development in a project or organization. As was learned earlier, many of these agile practices are found in standard agile methods such as Scrum, Extreme Programming, and Kanban. Some practices have also emerged without being related to a specific agile method, such as Continuous Deployment which is soon explained. In a scientific survey on the usage of agile methods in the Finnish software industry, which is also considered a reliable survey (Stavru, 2014), the following list were the most widely used agile practices (Rodríguez, Markkula, et al., 2012):

1. Prioritized work list
2. Iteration/sprint planning

3. Daily stand-up meetings
4. Unit testing
5. Release planning
6. Active customer participation
7. Self-organizing teams (i.e. self-managing teams or autonomous teams)
8. Frequent and incremental delivery of working software
9. Automated builds
10. Continuous integration
11. Test-driven development (TDD)
12. Retrospectives
13. Burn-down charts
14. Pair programming
15. Refactoring
16. Collective code ownership

It may be worth noting that Rodríguez, Markkula, et al. (2012) suggest no one-to-one relationship between these practices and agile methods, and do not describe them in any particular detail. For example, prioritized work list, daily stand-up meetings, and retrospectives may be used as part of several methods, but may be implemented differently. The list is still a useful overview of various widely used agile practices without specifying implementation details.

Furthermore, I will present two practices that are relevant to this study, but have not yet been properly described in the included literature on agile methods. The first one is *self-organizing teams*, which is also found in the above list of most widely used practices. The second one is *Continuous Deployment*, which is a practice that is not in the top list, but is important in DevOps and in recent trends in software development.

Self-organizing teams

Agile software development focus less on teams consisting of specialized roles, and focus more on self-organizing teams that encourage role interchangeability (Nerur, Mahapatra, et al., 2005). Terms like self-managing and autonomous teams is used interchangeably for this practice. The practice involves giving teams the authority and capability to effectively organize and manage their own work. According to Moe et al.

(2009), self-managing teams offer potential advantages over traditionally managed teams as they bring decision-making authority to the level of operational problems and uncertainties, and thus increase the speed and accuracy of problem solving. The practice have been valued in agile methods, such as Scrum and XP (Section 2.2.2), for these reasons and is also emphasized in the agile manifesto (Fowler and Highsmith, 2001). The performance of self-managing teams depends not only on the team's competence in managing and executing its work but requires management to facilitate an appropriate organizational context. Fully self-managing teams consist of cross-trained generalists who can take on several different tasks, which are collocated, and who build trust and commitment (Moe et al., 2009).

Continuous Deployment

In the section about XP (2.2.2), Continuous Integration (CI) was listed as one of its practices. While Continuous Integration mainly focuses on automation of the code integration and build process, Continuous Deployment is a logical extension. Continuous Deployment entails automating the entire workflow all the way from integrating and building good code to deploying it to to some environment such as production, but not necessarily to actual users (Fitzgerald and Stol, 2014). The goal is to reduce the time and resources it takes to deploy new software features or changes, and consequently lead to more rapid releases of software (Rodríguez, Lwakatare, et al., 2017). Continuous Deployment, and thus also Continuous Integration, are practices that are often used in conjunction with DevOps, which we previously learned (Section 2.2.3).

2.3 Method Tailoring

This section will present a background on method tailoring based on relevant scientific references. First, an introduction to method tailoring theory is given. Then, different approaches to method tailoring will be described as they appear in literature.

2.3.1 Introduction to Method Tailoring Theory

Both practitioners and researchers are continuously looking for new and better ways to develop software. As new methods arrive, practitioners tend to adopt them in hope that they will solve their own software development challenges (Brooks, 1987; Conboy and Fitzgerald, 2010). Brooks (1987) highlighted a problem with this. He argue that the constant strive for new and simplified models that can be reused to reliably develop complex software across projects is misguided, saying "there is no silver bullet". He argued that software is fundamentally challenging because of its inherent properties; *complexity*, *conformity*, *changeability*, and *invisibility*. As a result, software can often be more complex than any other human constructions. The complexity of software makes it difficult to maintain a complete overview which may lead to technical and managerial problems. Complexity also increases due to the need for conformity to different interfaces. Software is easily changeable, often easier than making changes in products found in construction and production. This inherent property can be a good thing, but also causes challenges. In construction and manufacturing you work with physical products, while software is invisible and difficult to visualize. Combined, these features make software challenging and full of changeable variables. This ultimately requires a lot of flexibility in the methods you use to create software (Brooks, 1987; Conboy and Fitzgerald, 2010).

As a result, methods are often not reused in its original textbook format in practice (Conboy and Fitzgerald, 2010). Instead, practitioners make their own variants that better suit their needs in different project contexts, often based on existing methods and practices (e.g., Fitzgerald, Hartnett, et al., 2006; Rolland et al., 2016). This has been researched in different software development contexts, and consequently different terms have been used. In Section 1.2.1, I concluded that the most commonly used term among researchers seems to be *method tailoring*. In Section 1.2.1, method tailoring was defined as "*a process or capability in which human agents through responsive changes in, and dynamic interplays between, contexts, intentions, and method fragments determine a system development approach for a specific project situation*". Other terms such as *method adaptation*, *context-based method use*, *method assembly*, and *method configuration* have also been used in research. Common to this research is that it focuses much on identifying possible approaches to method tailoring which may be used to tailor methods in practice. Many of the approaches are similar, and Fitzgerald, Hartnett, et al. (2006) among others argue that these can be categorized within two overarching approaches to method tailoring; *method engineering* and *contingency factor approaches*. These will be presented in more detail in the next section 2.3.2.

Conboy and Fitzgerald (2010) argued that method tailoring is particularly important for agile methods for several reasons. Agile software development introduce new issues and complexities through its values and principles such as "individuals and interactions over processes and tools". In this case, they argue that agile practices needs to be much more focused on elements such as personal characteristics and team dynamics. Another example they made, is the emphasis on close collaboration with stakeholders. This puts more pressure on tailoring to effectively support interaction with stakeholders and complexities within their organizations. While agile methods are in most cases an improvement upon traditional methods, the vast set of values and principles induce increased complexity and problems that makes adoption and tailoring of agile methods challenging.

Another argument for the importance of agile method tailoring lies in the definition of *agility*. This was briefly brought up when presenting agile principles and defining *agility* in Section 2.1.2. The goal of agile methods is to assist developers with creating high-quality software in an agile manner. Agile methods are supposed to be 'just enough' methods as they seek to avoid prescribing cumbersome and time-consuming processes that add little value to the final product and elongate the development process (Fitzgerald, Hartnett, et al., 2006; Highsmith and Cockburn, 2001). It revolves a lot about being able to change, not only the product, but also the way you work (Conboy, 2009). As a consequence, researchers suggests that agility must also be incorporated into the methods by deliberately designing lightweight agile methods that are amenable to tailoring (Abrahamsson, Warsta, et al., 2003; Henderson-Sellers and Serour, 2005; Turk et al., 2002). Software development methods that are very prescriptive in nature are argued to be less adaptive and amenable to tailoring. A method is prescriptive if it require specific "rules" to be followed such as "the projects always use timeboxed incremental development cycles with a maximum increment length of two to four weeks". While many methods suggest such rules, some original literature on certain methods also acknowledge and expresses that the rules should not always be followed as-is in any situation. Some literature also provide guidance on how the method can be tailored. When this is the case, the method can be regarded as amenable to tailoring even though it is prescriptive in the first place (Abrahamsson, Warsta, et al., 2003).

Kniberg and Skarin (2010) argue that the degree to which a method is adaptive to a situation depends on the number of rules present in the method. By "rules" they refer to parts and practices of the method such as *artifacts*, *roles*, and *activities*. A large amount of strict rules gives many constraints, and thus fewer options open. Agile methods are generally regarded as *lightweight* methods that contain few rules. For example, one important value in agile development is "individuals and interactions over processes and tools" (Fowler and Highsmith, 2001). However, some agile methods are more prescriptive than others. Traditional software development methods are comparatively even more prescriptive than agile methods, which makes them less amenable to tailoring. Kanban (Section 2.2.3) only contain three rules which makes it more amenable to tailoring. Abrahamsson, Warsta, et al. (2003) argue that Scrum and XP (Section 2.2.2) are also amenable to tailoring. Not only because they contain relatively few rules, but because they don not force all the rules and acknowledge that tailoring might be needed in their original literature. Other research also demonstrate that methods such as Scrum, XP, and Kanban have been tailored in various ways in software development projects (e.g. Conboy and Fitzgerald, 2010; Corona and Pani, 2012; Fitzgerald, Hartnett, et al., 2006). Kniberg and Skarin (2010) illustrate the degree of adaptability of some known methods as shown in Figure 2.6. As illustrated, they argue that a "Do Whatever"-method that prescribe no rules is the most adaptive in comparison.

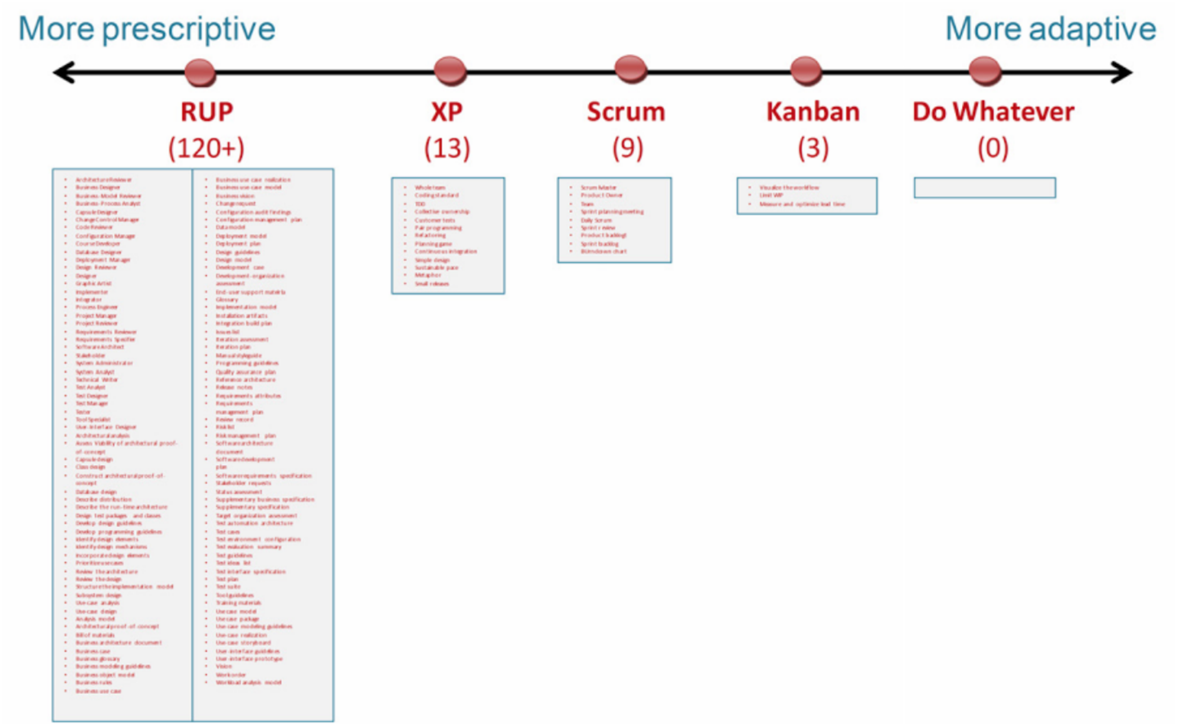


Figure 2.6.: Degree of adaptability of agile software development methods (Kniberg and Skarin, 2010)

2.3.2 Method Tailoring Approaches

Different approaches to tailoring software development methods exist. Researchers have identified approaches from practice and suggested new ones based on research. Research on method tailoring existed prior to agile software development. However, with the paradigm shift, new research explored method tailoring approaches also in the context of agile methods. In fact, the interest for method tailoring have steadily increased following the release of the agile manifesto (Campanelli and Parreiras, 2015). Since then, researchers have applied existing approaches also in context of agile methods (e.g., Conboy and Fitzgerald, 2010; Fitzgerald, Hartnett, et al., 2006).

There are mainly two overarching approaches or categories of method tailoring identified in research, namely *contingency factor approaches* and *method engineering* (Campanelli and Parreiras, 2015). However, Campanelli and Parreiras (2015) found in their recent comprehensive literature review on the topic that some research papers did not classify or explain the approach used. Thus, it is unclear whether those could be categorized into the two main overarching approaches or not. The two overarching approaches will be described in the next two subsections.

Pedreira et al. (2007) differentiate between formal and informal approaches to tailoring. They found that most research papers propose some formal approach, rules or guidelines to systematically tailor a software process. This formality is also evidenced in the two overarching approaches to be described in the next subsections. Moreover, they also found that tailoring is often performed in an informal manner in practice (i.e. little or no formal process, rules or guidelines for tailoring). Large organizations had a tendency to take on more formal approaches to tailoring, while in smaller organizations or projects it seemed more practical with informal tailoring. This have been observed in other research as well, where it has been described as ad hoc and unstructured tailoring approaches (Conboy and Fitzgerald, 2010). In other words, practitioners often do not follow formal and prescriptive approaches as proposed in method tailoring research. Some have argued that this may be problematic as it may potentially cause suboptimal results, or that little is learned about tailoring efforts across projects (Conboy and Fitzgerald, 2010). Moreover, Patel et al. (2004) found that teams in practice often tailored methods based on past experiences. This was also found in the case study by Fitzgerald, Hartnett, et al. (2006) where an informal approach was used. One benefit of a more systematic, formal and less ad hoc approach to tailoring is that the result would not depend so much on skills and past experiences (Pedreira et al., 2007).

Contingency factor approaches

Contingency factor approaches, also called contingency-based method selection (Conboy and Fitzgerald, 2010), is a method tailoring strategy that involves selecting a specific existing method that is best suit for a particular context (Iivari, 1989). The approach is based on the premise that no method is universally applicable; there is no silver bullet. Central to this approach is that teams or organizations that wishes to apply it must be experienced with multiple different methods. They must also be able to evaluate which method is best suited for the given context based on the contingencies of the situation. For this, various criteria for tailoring can be used for evaluation. Such criteria may be based on influencing factors such as team size and team distribution, external factors such as contract type and client availability, and a multitude of other factors that influence a software development process (Campanelli and Parreiras, 2015; Kalus and Kuhrmann, 2013).

A contingency factor approach is argued to be a challenging approach in practice, and contain several flaws. For one, extensive experience and knowledge on multiple or even one method have seemingly been rare among developers in practice. When this is the case, it may negatively affect the effectiveness of tailoring efforts (Conboy and Fitzgerald, 2010). Overall, it is thought that achieving much competence with many methods is unlikely to be the case for many. The second major flaw with contingency factor approaches is that they assume that some specific existing method can cover all the needs and contingencies of a situation. In other words; if you have many enough variants of agile methods, you will find one that suits your particular project (Iivari, 1989; Kumar and Welke, 1992). Furthermore, contingency factor approaches are not concerned with adapting existing methods or generating new ones. These aspects and assumptions are arguably contradictory with a core premise of method tailoring in the first place; that all project contexts are different and that no predefined method can be applied to cover all needs of any given project.

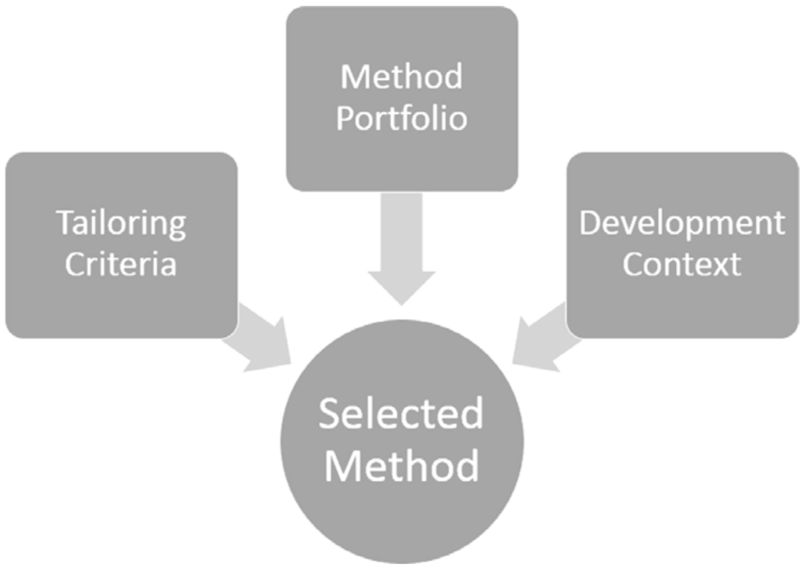


Figure 2.7.: Contingency factors process for method selection (Campanelli and Parreiras, 2015)

Method Engineering

Method engineering approaches proposes that methods should be created or "engineered" to be applied to specific development situations. A method engineering approach acknowledges that a method to guide development is indeed advantageous in projects, but recognizes that no predefined method can cover all needs (Harmsen et al., 1994).

Method engineering involves creating situation specific methods from pre-defined and pre-tested method fragments (Kumar and Welke, 1992). A method fragment can be a description of a software development method, or any coherent part thereof. It can be principles, fundamental concepts, products to be delivered, development activities, techniques, tools, etc. (Aydin et al., 2004). As a result, method engineering can be described as a meta-method process (Kumar and Welke, 1992). The approach can be used to create methods by mixing and matching method fragments into a more suitable method. Campanelli and Parreiras (2015) found that method engineering was the most used tailoring approach by a large margin based on the existing catalog of research. The research suggests that the selection and combination of method fragments requires a fragment repository and a dedicated method engineer to handle the repository in order to be operational (Fitzgerald, Russo, et al., 2000). In association to this, a common recommendation in research is also to use specialized computer-aided tools to support the method engineering process (Brinkkemper, 1996; Henderson-Sellers and Ralyte, 2010). Figure 2.8 illustrates a method engineering process from one of the initial works on method engineering (Brinkkemper, 1996).

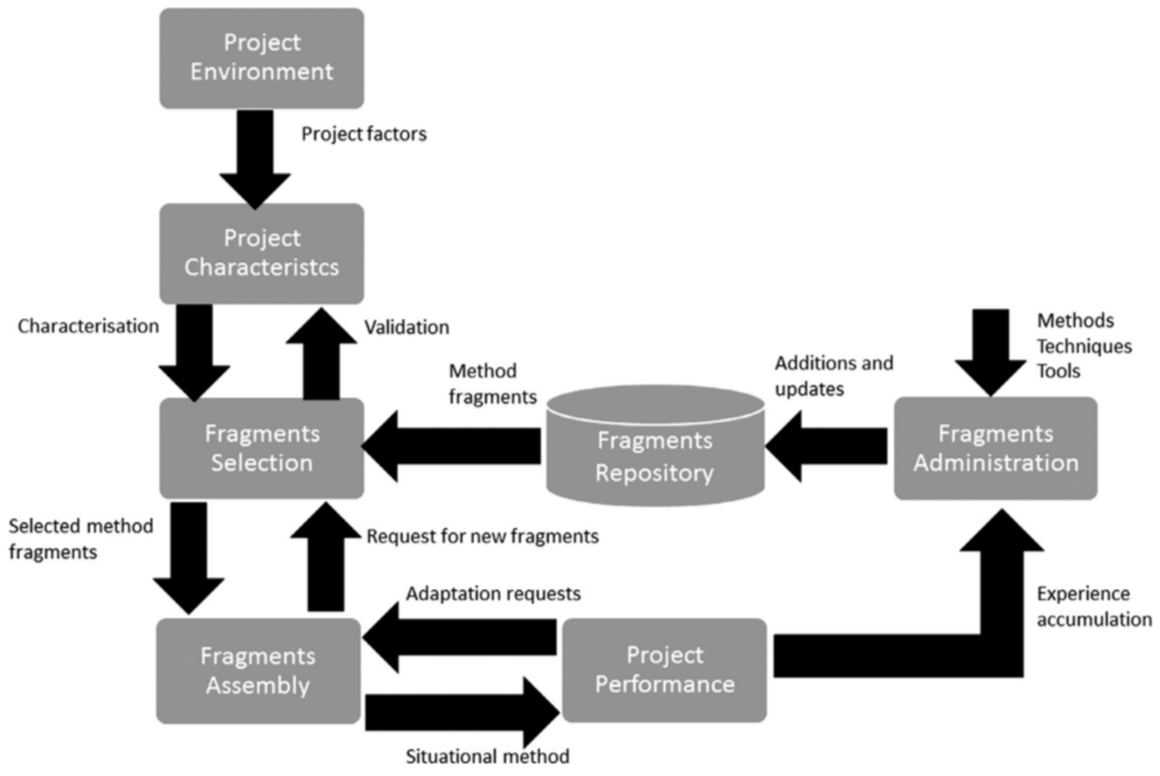


Figure 2.8.: Method engineering process for method definition (From Campanelli and Parreiras, 2015 based on Brinkkemper, 1996)

Method engineering, including contingency factor approaches, have generally been portrayed as very formal and structured forms of method tailoring that have been based on theoretical and conceptual arguments. Fitzgerald, Hartnett, et al. (2006, p. 203-204) describe this as a marked feature of method tailoring research: "*both the contingency and method engineering research is that they are largely deductive in nature and employ theoretical and conceptual arguments to support how methods should be tailored or constructed*". As is illustrated in Figure 2.8, method engineering is a comprehensive and very defined process. Because of this, it has received some skepticism regarding practical applications and whether such a process is actually welcome in many organizations or projects. The high degree of formality and resources required to implement and use the proposed approaches are among the biggest challenges and potential problems with these from a practical perspective (Campanelli and Parreiras, 2015; Fitzgerald, Hartnett, et al., 2006).

Fitzgerald, Hartnett, et al. (2006) found through case study research that a similar approach to method engineering may be successfully and purposefully done without having dedicated method engineers or a repository of method fragments. In their case study, practitioners had purposefully selected various practices from the agile methods Scrum and XP specifically and combined them into their own tailored method with satisfactory results. They also found that the tailoring was to a large extent done based on previous experience and understanding of the individual agile practices. Similarly, Conboy and Fitzgerald (2010) studied how practitioners tailored XP by selecting suitable XP practices for different situations. These approaches shared the same fundamental philosophy as method engineering, where suitable methods were created based on a selection of practices, but in a much less formal and structured manner.

2.4 Summary

This chapter has provided a thorough review of important concepts related to agile software development and method tailoring.

The initial section included a comparison with traditional methods and a presentation of both benefits and challenges related to agile software development. This was followed by a more in-depth look on the concept of agility and what it means for a software project method to be agile. Thereafter, the initial section ended with a brief introduction to the recent trends towards continuous software delivery. These initial aspects provides a valuable background and perspective for later discussion of findings in this study.

The second section proceeded with learning about both iterative and flow-based agile methods and an overview of widely used agile practices. This background will be useful for understanding the agile methods and practices used in the case to be studied, and how these were tailored compared to methods and practices in literature.

The third section introduced method tailoring and motivations for why tailoring of agile methods is necessary and important. The section also introduced concepts such as method adaptability, and characteristics of proposed tailoring approaches such as the contingency factors and method engineering approaches. The literature on method tailoring is central to this study, and will prove valuable for evaluating and discussing the approach to method tailoring in the case.

Method 3

The method chapter will present the guidelines and approach used for conducting this research. The first section will give an overview on how the literature review was performed. Furthermore, case study will be introduced as the chosen research strategy. This includes describing key research procedures, such as case selection, data collection, and data analysis. Thereafter, some clarifications regarding the reporting of the results and confidentiality is provided. The last section of the chapter presents the main tactics that were used to improve the validity of the research. Justifications for the choices made will be provided when appropriate.

3.1 Literature Review

A literature review was a key part of this study. The literature review was performed in order to explore possible and relevant research topics, to get a grip on existing research related to the chosen topic, and to form the theoretical basis for the study. Reviewing literature was an ongoing activity from start to finish.

The overall strategy for searching for literature was inspired by the structured approach suggested by Webster and Watson (2002) summarized as follows:

1. Use databases such as Web of Science[1] to accelerate identification of relevant articles using selected keywords. Since the keywords are unlikely to catch all relevant articles, scanning through leading journals where major contributions are likely to be found is a useful way to catch others you have missed. Since software engineering is a multi-disciplinary field of research, it may also be necessary to look for journals in other related disciplines.
2. Go backwards by reviewing the citations for the articles that were identified in step 1) to determine other articles to consider.
3. Go forward by using databases such as Web of Science to find the articles identified in the previous steps and evaluate whether they are relevant for the review.

[1]https://webofknowledge.com/

Several scientific databases were used for literature search. A major part of the literature was identified using deliberately selected search queries related to key concepts of the research. Much literature was also discovered using a "snowball" strategy by reviewing the references of key articles, as suggested in the approach by Webster and Watson (2002). For practical reasons I did not spend much time scanning through complete journals, but rather the reference lists of individual journal articles. In addition, some very interesting articles were identified through article suggestion services supported in search engines such as Elsevier's database ScienceDirect[2]. The service suggests similar or related articles based on the ones you visit. The databases used to find literature are listed in Table 3.1.

Database	**Description**	**Location**
Science Direct	Leading citation indexing service for scientific, technical, and medical research	http://www.sciencedirect.com/
Google Scholar	Free search engine which indexes full-text or metadata for academic literature across disciplines	https://scholar.google.no/
Oria	Citation indexing service provided at Norwegian universities which indexes academic literature across disciplines	https://oria.no/
Web of Science	Scientific indexing service used for accessing leading scholary literature across disciplines	https://webofknowledge.com/

Table 3.1.: Databases used for literature search

3.1.1 Literature Inclusion and Exclusion Strategy

The process of deciding which literature to include or to exclude from the search results was an ongoing and iterative process. The initial evaluation of the literature was done by reading the abstracts to evaluate whether the literature concerned relevant topics and concepts. If it seemed relevant, I proceeded to do a more thorough evaluation by scanning through the introduction, and the conclusions to get a grip of the key findings. I primarily looked for three types of literature:

[2]http://www.sciencedirect.com/

- Literature that covered topics that could potentially be used as a theoretical basis to support data analysis and discussion of results. These could for example be research articles and systematic literature reviews on topics such as agile software development, agile methods and practices, and method tailoring.
- Literature that could be used to provide background and context for the study.
- Existing empirical research that could potentially be incorporated into the study later for comparison to my findings.
- Literature on writing and conducting research, in particular case study research. This was to be used for both guidance on how to perform case study research, and to underpin my research strategy choices which is to be presented further into this chapter.

Important criteria for including core literature were also related to the quality of the literature. A quality assessment was made for each article, book, etc. These are typically the criteria that were used:

- The literature had a relatively high number of citations, and thus was likely widely accepted by the research community.
- The literature itself used many references to support findings and the theme of the research.
- The literature was empirical research, such as case study, survey, or experiment. Secondary studies such as a literature reviews that drew upon other high quality literature was also highly regarded.
- The literature was published in journals, or in some cases conferences. Some other types of literature, such as books, were sometimes included if they were evaluated to be of high relevance and had relatively many citations.

Not all criteria had to be met at all times. For example, if a study was very recently published in a renowned journal by prominent researchers in the field, it would naturally still not have many citations as of yet. It would, however, likely still be of high quality because it met other criteria. In other cases articles from conference proceedings were used if they met other criteria and was regarded to be very interesting and relevant. Not all papers were empirical research. For example, books or literature explaining specific agile methods in general. In general, a combination of the above criteria and the overall impression of the literature was used to evaluate its quality and significance.

EndNote[3] was used for managing the bibliography. Scientific papers, including references, could be downloaded directly from the search engines on the web and loaded into EndNote. These were then categorized into groups based on the main concepts that each paper dealt with. Using EndNote, references could be easily exported to a LaTeX[4] bibliography format for citing while writing. Categorizing the articles made it easier to keep track of relevant articles for different concepts. Towards research completion, all included references were checked one-by-one for errors that might have occurred when downloading the citations pre-formatted from the web.

3.2 Case Study Research

The research methodology adopted for this study was an *exploratory single-case study*. In the background and motivation (Section 1.1), a motivation for more empirical research on agile software development and agile method tailoring was provided. In order to investigate these topics empirically, a qualitative approach was expedient. While a quantitative approach could be used, a qualitative approach generally allows for a more in-depth investigation of research questions due to factors such as more flexible data collection methods (Runeson and Höst, 2008).

The intent of this master thesis was to investigate how practitioners performed method tailoring in software development projects, which made case study an appropriate choice. Case study research has traditionally been widely used in social science. Runeson and Höst (2008) points out the relevance of case study as a research method also within the software engineering field. They highlight software engineering as a multi-disciplinary field with much in common with social science which often investigates individuals, groups, organizations, and political phenomena. This is similar to software engineering research, which mostly examines how software development, operation, and management are conducted by individuals, groups, and organizations in different circumstances (Runeson and Höst, 2008). Therefore, case study is also a good alternative for research in software engineering topics such as agile software development.

Case studies are normally distinguished between single-case and multiple-case design. Single-case focuses on studying one case, although with the possibility of incorporating other case studies for comparison purposes (Yin, 2014). Since the timeframe and the scope of this research was limited, studying a single case rather than multiple

[3]www.endnote.com/
[4]https://www.latex-project.org/

cases was considered appropriate. Some case study articles on agile method tailoring have been used to discuss and support findings. However, emphasis was not put on comprehensive comparison with other case studies. Instead, this study has focused on using the available theoretical basis as a whole to discuss and support findings. Runeson and Höst (2008) summarizes the characteristics of a case study as outlined in Table 3.2. Consequently, these are also typical characteristics of this research. A

Methodology	**Primary objective**	**Primary data**	**Design**
Case study	Exploratory	Qualitative	Flexible

Table 3.2.: Characteristics of the research methodology (Runeson and Höst, 2008)

case study uses a flexible design. This means that the parameters of the study, such as research questions and interview guides, may be changed during the study. This is opposed to a fixed research design, such as experiments or surveys, in which the parameters are set in stone early on. This flexibility was utilized, and allowed the study to emerge and evolve as the work went on. This was important, since little was known about the case beforehand. Ultimately, this enabled more maneuverability in the study as I learned more about the case and ability to go more in-depth on interesting topics as they emerged. In this regard, both the problem statement, research questions, and interview guides went through multiple iterations throughout the course of the study. Moreover, the primary data collected was qualitative. That is, the data took the form of words, descriptions, tables, figures, etc., and not statistical data (Runeson and Höst, 2008).

There are different types of case studies, such as exploratory, explanatory, and descriptive case studies. However, Runeson and Höst (2008, p. 135) points out that case study research have primarily been used for exploratory purposes, that is "finding out what is happening, seeking new insights, and generating new ideas and hypothesis for future research". When the studied case was initially identified, I learned that they had adopted a modern approach to agile software development which was arguably very interesting and relevant. However, it became apparent that little research had previously been conducted on method tailoring in similar contexts. Moreover, the conclusions of a recent systematic literature review on agile method tailoring argued that future work should empirically explore how agile methods can be tailored, with attention to tailoring approaches, in order to increase knowledge on the topic (Campanelli and Parreiras, 2015). Oates (2006) describe an exploratory study as a research strategy that can be used whenever there is limited support in the current literature on the topic of interest. Thus, an exploratory investigation of a real-world setting can be done in order to acquire new insight, as well as to generate ideas for future research. For these reasons, an exploratory case study was appropriate for this study.

Runeson and Höst (2008) outline the the main steps in a case study research process as follows:

1. Case study design: objectives are defined and the case study is planned
2. Preparation for data collection: procedures and protocols for data collection are defined
3. Collecting evidence: execution with data collection on the studied case
4. Analysis of the collected data
5. Reporting

This also describes the overall process that was adopted for this case study. It is worth noting that this was conducted as an iterative process, where each step was done repeatedly during the course of the study. According to Yin (2014), an iterative procedure is recommended. The details on how each step and underlying tasks was conducted is described in the upcoming sections.

3.3 Case Selection

For case studies, the case and units of analysis should be selected intentionally to ensure it is appropriate for what you want to study (Runeson and Höst, 2008). I had a predetermined desire to study tailoring of agile methods, and therefore spent much time initially looking for a project that seemed interesting and relevant to this. Within practical research, one limitation is that a case is selected based on availability (Runeson and Höst, 2008). The case choice in this study was primarily based on availability, but some time was spent initially to identify different case alternatives and select one good candidate. Three software development projects were identified and evaluated through conversations with key representatives from each project. A summary of the criteria used to assess each case was:

- Agile methods and practices were used.
- The methods and practices were more or less tailored to the project context.
- The project should already have been ongoing for a good amount of time to ensure that the project had a history, and that the participants had a reasonable amount of experience from the case.

- It was possible to interview multiple people in different roles that had experience working within the project to ensure that multiple viewpoints were covered.
- The interviews could be conducted in two rounds and last 30 minutes or more to have time to dive into both broad and in-depth questions.

These criteria were required for the project to be accepted as a candidate. The selected case for this study was particularly interesting and stood out among the other candidates while also fulfilling the criteria.

Yin (2014) argue that for single-case design, a case can be selected that represents a:

- critical test of existing theory
- extreme or unusual circumstance
- common case
- a revelatory or longitudinal purpose

I would argue that this case represents an *extreme circumstance*. I found very few similar cases in method tailoring research or other studies in general. Also, the project clearly represented a rather modern agile context compared to many others. The implications of this was that I was able to study agile method tailoring in a rather new and extreme circumstance, which was an excellent opportunity for gaining new insight and also very interesting.

3.4 Preparation for Data Collection

For case study research it is recommended to maintain a case study protocol that can be used as a plan and a guide for carrying out data collection and analysis work. According to Runeson and Höst (2008), a case study protocol defines the detailed procedures for collection and analysis of the raw data. This has advantages such as to not forget to collect data as intended, and to get a picture of what is required to achieve data collection and analysis goals. During my work, I maintained a repository of documents that contained plans for conducting interviews, a schedule for the overall research, interview guides, information about informants, observation notes, formal agreements and more. In addition, quite a bit of information about practicalities was contained in e-mails. Although it might have been practical to collect everything in one study protocol document, mine was more informal and consisted of a number of documents. Overall, this enabled me to keep track of the research work throughout, and to better prepare for data collection procedures. Most of these documents have not been included as appendices in this thesis to avoid leaking confidential information.

An important part of the preparation for data collection was to make sure that the data to be collected was as high quality and rich as possible. There are multiple ways to contribute to this, such as through triangulation strategies. Triangulation is when you take different angles towards the studied object, with the intent to get a broader picture of the case in study. In a case study, which is primarily qualitative in nature, the quality of the study can be increased through triangulation (Runeson and Höst, 2008). Four types of triangulation may be applied to case study research:

- Data (source) triangulation - where you use more than one source for data collection, or collect the data at different occasions.
- Observer triangulation - where you use more than one observer in the study.
- Methodological triangulation - where you combine different types of data collection methods, e.g. qualitative and quantitative methods.
- Theory triangulation - where you use alternative theories or viewpoints.

This study primarily used data (source) triangulation. One importance in data collection in research is to use multiple sources for collecting data. This can be done to limit the effects of one interpretation of one single data source (Runeson and Höst, 2008). If a conclusion can be drawn from several sources of information, the conclusion is stronger than if the conclusion was based on a single source. This was important during my data analysis, where I tried to draw conclusions based on the same or similar statements from multiple informants whenever possible.

Because I was one person alone doing the data collection, observer triangulation was not used. Methodological triangulation could have been used, but due to limited availability of informants and the time schedule, I only focused on qualitative data collection through interviews. I did use a form that was answered by some informants during the interviews, but this was only intended as a starting point for qualitative data collection. More on this in Section 4.2.2. I would argue that theory triangulation was not used to any particular extent, although theory and literature was used to discuss findings. This study was mainly exploratory and used an inductive approach where I explored the case without strictly adhering to theories as an analysis lens.

3.5 Data Collection

For case-study research, one of the most important sources of evidence is the interview (Yin, 2014). While surveys and experiments are particularly good for quantitative research using more fixed research designs, interviews are often used in qualitative research because rich data can be collected through guided and more in-depth conversations. Runeson and Höst (2008) refer that data collection methods may be divided into three levels:

- First degree: Direct methods, such as interviews and questionnaires where the researcher is in direct contact with the participants.
- Second degree: Indirect methods, where the researcher collects data, but not in direct contact with the participants. For example observations, shadowing, and video recording of the work place.
- Third degree: Study and analysis of existing work artifacts, such as documents, requirement specifications, and reports.

The first and second degree methods are generally considered advantageous in case-study research because it allows the researcher to decide what data is collected and in what format. Third degree methods serve as a good supplement, but does not allow for the same amount of control (Runeson and Höst, 2008). All three degrees of data collection methods were used to varying extents. The choices will be further described in the following sections about *Semi-Structured Interviews* and *Observations and Document Analysis*.

3.5.1 Semi-Structured Interviews

Semi-structured interviews were used as the primary method for collecting evidence. One reason for this choice was that few details about the case was known beforehand, which made it challenging to know exactly what was relevant to ask up-front. Semi-structured interviews allowed for some improvisation and exploration both during and across interviews through both open and closed questions (Runeson and Höst, 2008). Questions were planned up-front, but they could be modified and asked in a different order depending on how the conversation developed. For these reasons, this was a good choice for the primary data collection method for this case-study.

The interviews were conducted in two rounds. This was necessary because of the initial lack of knowledge about the case of study, and for data triangulation. The goal of the first round was therefore to get to know the people that worked in the project, to get a good overall understanding of the case, and begin to ask some more focused questions towards topics such as method tailoring. The first round was absolutely necessary to uncover potential directions for further questions, and to minimize the risk that important information was missed in the last round. The second round of interviews was characterized by more closed questions and by going more in-depth on topics of interest based on the analysis of the initially collected data.

All of the interviews in both rounds were relatively long, ranging between 40 minutes to more than 1 hour in duration. This resulted in a large amount of material to be analyzed. A total amount of 131 pages were transcribed from audio recordings of the interviews. More on the transcription process in Section 3.6.

Average interview duration	**Total transcribed material**
54 minutes	131 pages

Table 3.3.: Key figures related to interviews and resulting material

An overview of the interview participants based on roles, and the total number of interviews conducted for each role is shown in the Table 3.4. While some informants were interviewed once, others were interviewed two times. This was mostly a result of limited availability of informants, since the interviews were conducted during busy work hours. People in a variety of roles were interviewed. This ensured that data was collected from subjects with a variety of different viewpoints, and thus contributed positively to data triangulation as mentioned in Section 3.4. A total of nine interviews were conducted, and a total number of six informants participated. Additionally, I was also able to talk informally with more people from the project during observations of project activities, in the hallways and during lunch. Notable observations that were made during informal conversations were written down as notes. The interview guides used during interviews have been enclosed in Appendix A.1.

3.5.2 Observations and Document Analysis

In addition to interviews, both observations and document analysis were done. Observations can be done to investigate how software engineers perform certain tasks that can be of interest in the study (Runeson and Höst, 2008). I was able to participate and observe a total of four stand-up meetings while visiting the project premises. I

Role	Number of informants	Number of interviews
Designer	1	1
Developer	3	5
Interaction designer	1	1
Project manager	1	2
Sum	**6**	**9**

Table 3.4.: Overview of the roles of informants and the number of informants and interviews for each role

was also able to observe other practices such as how the teams organized themselves in the work place, how and where they held their meetings, how they used important tools, and how the they involved and coordinated work with their customers.

Some existing documentation was also collected. This was very limited in comparison to the interview material, and consisted mainly of two presentations containing general information about the project as well as a few web pages. This was a useful source for getting an initial understanding of the case and for retrieving illustrations which could be included in the results. Some drawings were also done by the informants during the interviews to illustrate concepts while talking which was also documented.

3.6 Transcription of Recorded Interviews

During the interview sessions it is recommended to record the discussion in a suitable audio or video format (Runeson and Höst, 2008). I chose to do audio recordings for my interview sessions. Thus, I could focus on the conversation and questioning.

The audio recordings were transcribed after each round of interviews. The tool Transcribe[5] by Wreally was used to transcribe the recorded audio to textual format. Audio transcription was a very time-consuming process. Transcribe, however, made the process less demanding by enabling automated pausing, rewinding, and resuming of the audio in configurable time intervals. Moreover, the conversations were transcribed in a manner that preserved as much information as possible as it was converted to text. For example, longer pauses and hesitation in sentences, corrections by the informants,

[5]https://transcribe.wreally.com/

and so on was preserved through structured formatting. This ultimately gave more context to the textual data, and more information to base the analysis on.

3.7 Coding and Data Analysis

The last and most essential step before reporting the actual findings were analysis of the textual data. Data analysis involves examining, categorizing, tabulating, testing, or otherwise recombining evidence, to produce empirically based findings (Yin, 2014).

When dealing with large amounts of qualitative data in a textual format it was useful to use NVivo[6], which is a tool specialized for doing qualitative data analysis. NVivo supports importing textual data, searching, coding and queries, adding memos as you work through the data, as well as functionality such as generating diagrams and charts. This allows for a structured and convenient approach to analyzing qualitative data, with many advantages over a more traditional "pen-and-paper" approach. The main activity performed in NVivo was coding the textual data. The adopted approach to coding was based on tutorials from NVivo, and can be summarized as follows:

1. Read through the textual data and identify concepts that could be of interest and relevance to the research context and research questions (an inductive strategy).
2. Create nodes (i.e. categories) in NVivo for the identified concepts, and code (i.e. categorize) the textual data to the appropriate node. The data should be coded for broad concepts, not specific ideas.
3. Code similar ideas together, and avoid a deep and complex node hierarchy.
4. Use searching and queries in NVivo to identify emerging themes and patterns in the data.

Moreover, emerging themes and patterns and the underlying coded data were further manually examined. This was then interpreted and validated manually. Keyword queries in NVivo made it easy to go back and confirm findings if there were any uncertainties. The node hierarchy used for coding is illustrated in Figure 3.1.

The strategy for analysis was an inductive approach. That is, instead of starting the analysis from a theoretical standpoint (deductive), the analysis was performed ground up by working with the data to identify relevant concepts. Yin (2014) recommend this approach as one of four possible approaches to case study analysis. An inductive

[6]http://www.qsrinternational.com/nvivo/what-is-nvivo

approach values freedom in exploring the case for interesting concepts relevant to the research questions, rather than restricting oneself to a set of theoretical propositions (Oates, 2006; Yin, 2014). Two potential pitfalls with this strategy is (a) if the researcher has insufficient knowledge of the field of study, relevant concepts may be hard to identify (Yin, 2014), and (b) the analysis is to some degree susceptible to subjective bias due to the researchers previous experiences, learning, and prejudices (Oates, 2006). In order to mitigate adverse results (a) much time was spent reviewing literature in the area of study to improve my ability to identify concepts of interest in the data, and (b) the data was frequently revisited to ensure that support for inferences existed in the data. Data analysis was performed after the transcription and anonymization of the first round of interviews were complete, and then again after the second round of interviews.

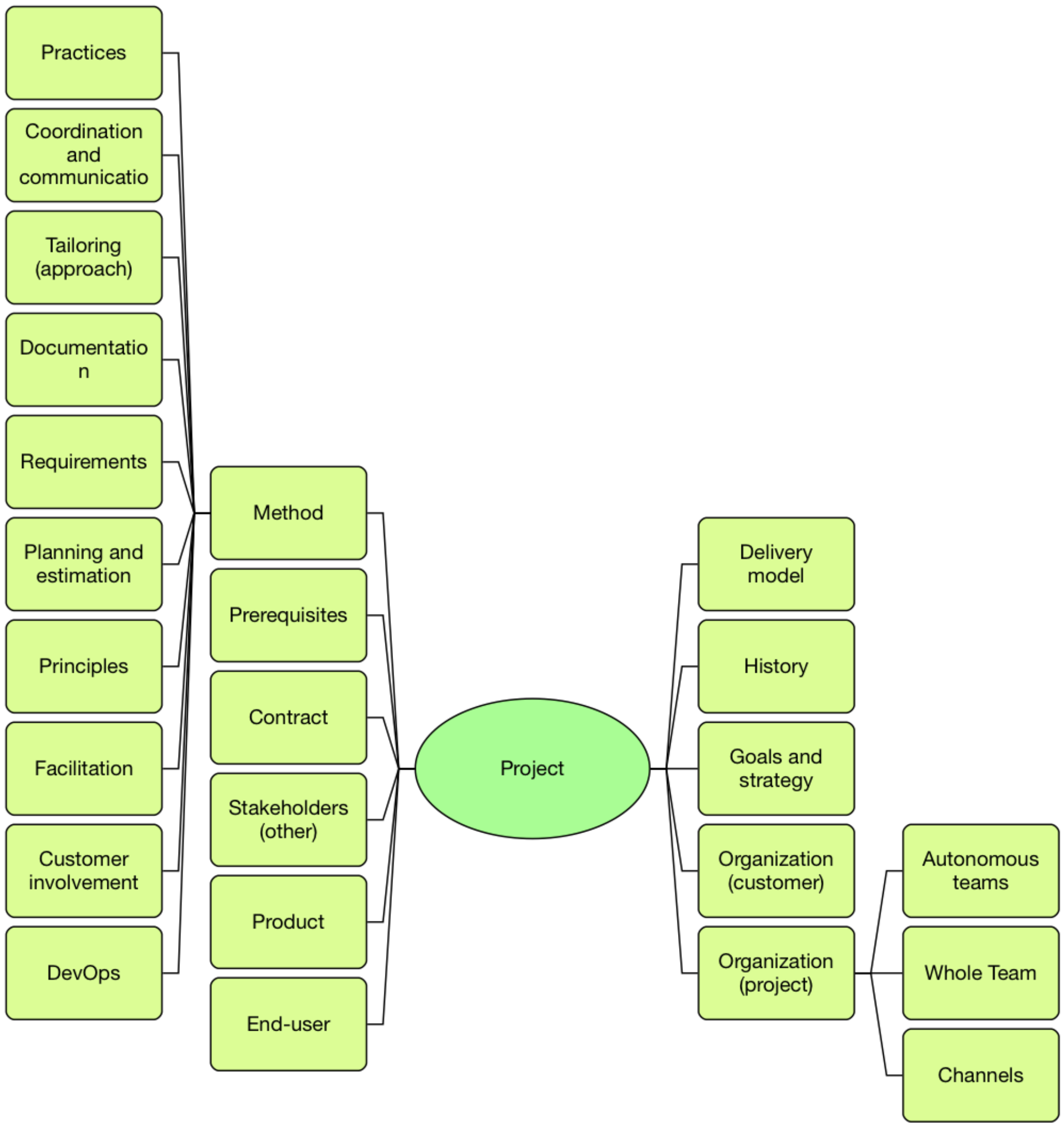


Figure 3.1.: Node hierarchy used for coding the textual data

3.8 Reporting of Results

Reporting the results involved systematizing and presenting the key findings from the analysis. One important clarification regarding the reporting of the results, is that the data was collected in a project in Norway and the interviews were conducted in Norwegian. Because the selected language for this master thesis was English, the results had to be translated from Norwegian which was not always a straight forward process. The results in Chapter 4 contain a high number of quotes from the interviews which was also translated. These were carefully translated to the best of my ability to not deviate from the original statements in Norwegian.

3.9 Confidentiality and Anonymization

When conducting a case study and dealing with data from real-world organizations, ethical considerations are important. At design time, it is therefore necessary to assess whether the information you will be dealing with is confidential information or not. Thus, it is important that the researcher and the organization explicitly agree on participation in the research and how the collected data is managed. Consent agreements through contracts are a preferred way to handle this (Runeson and Höst, 2008). A cooperative agreement and a confidentiality agreement was signed by both parties prior to the data collection to comply to these aspects. The confidentiality agreement (unsigned) is attached in Appendix A.3.

Every informant was made aware of what the data would be used for, and how I intended to use the data. Informants were also offered the opportunity to review the transcribed material or the results from the analysis prior to publication. However, it was stated that this was not necessary as long as the data was properly anonymized. This implied that the data should be anonymized in such a way that the people, the supplier and the customer organization was not revealed in the final results. This included that the data was not publicly released in any way.

As the data was transcribed, it was immediately anonymized by removing names of companies, people, products and services, stakeholders, and other information that was revealing. This made it more convenient when actually doing the analysis and presenting the final results. All results from the studied case have been anonymized in order to comply to confidentiality. This should not have any meaningful implications on the results. The results were presented in such a manner that context was preserved as much as possible without revealing identities.

3.10 Research validity, reliability and generalizability

This section mainly focuses on the tactics used to improve validity in the study, which also includes reliability and generalizability. Assessment of the actual limitations of the study which may have impacted validity is presented in the end of the discussion chapter (Section 5.3).

An important feature of a research design in general is that it is based on a logical set of statements. The systematic and logical nature of research is what makes research trustworthy and contributes to increased knowledge. However, a number of pitfalls may contribute to decreasing the overall quality of research if not taken into consideration in the research design (Yin, 2014). One important quality attribute of a study is validity, which denotes its trustworthiness and the extent to which the results are true and not biased by a researcher's subjective point of view (Runeson and Höst, 2008). Validity must therefore be considered from the beginning of a study. However, it is important to evaluate validity and whether there are any potential threats to validity also after the analysis phase. Yin (2014) name four tests of validity and corresponding tactics that may be used. These are also explained by Runeson and Höst (2008). In the following, these will be described and which tactics was used to improve validity in this research.

Construct validity

Reflects on whether the studied operational measures are actually what the researcher had in mind, and what is investigated according to the research questions. For this, using triangulation is here a possible tactic (Runeson and Höst, 2008; Yin, 2014). Data triangulation was used in this study to improve construct validity, which was explained in Section 3.4. Other forms of triangulation could potentially have been used to further improve validity.

Internal validity

Concerns causal relationships and inference. Causal relationships are important in explanatory case studies, and not of particular concern in exploratory case studies (Yin, 2014). The second issue is regarding making inferences. The analysis of this exploratory study was to a large extent about making inferences from the gathered data. To ensure as good quality inferences as possible, I always tried to base these on statements and patterns in the data from multiple sources. Since the number of informants were relatively few inferences were in some cases based on one informant's statements, but the general understanding of the case and related statements by other informants was used for support. However, this may be considered a weakness in the research method, and could have been improved by interviewing more informants.

External validity (generalizability)

Concerns whether it is possible to generalize from the findings, and whether the results are of relevance for other cases. Generalizing from case studies beyond the immediate study is often regarded as misguided and have received some criticism over the past. Case study research focus in-depth on specific situations and contexts which limits the possibility for generalizing to other settings beyond the one that is actually studied (Lee and Baskerville, 2003). Flyvbjerg (2006) stresses that this does however not make case studies unimportant, but is necessary and valuable in line with other methods because it provides in-depth insight in way that some other methods do not. With this in mind, I was careful not be too bold and make generalized conclusions beyond this study. However, one way to support external validity for case studies is to generalize to theory (Yin, 2014). Therefore, I used existing theory to discuss findings and for support when drawing conclusions.

The reader should be aware that some results from this study may not be generalizable and thus not necessarily applicable to any given context. However, it is likely that many of the findings may still be valuable and applicable to other projects in practice. In fact, many of the findings regarding method tailoring and agile practices may be applicable also to more general settings other than those who pursue continuous software delivery specifically. Thus, the findings may very well be of interest to other researchers and practitioners for use also in other contexts beyond this studied case.

Reliability

Concerns the extent to which the analysis is dependent on the specific researcher. That is, whether the results would be the same if another researcher conducted the same study later on. To address this, I have documented the research process through various documents in what may be regarded as an informal case study protocol (Section 3.4). This could potentially have been reused by another researcher and hopefully arrive at the same results. This is recommended as a tactic to improve reliability by Yin (2014). Part of the case study protocol was the interview guides, which are attached in Appendix A.1.

Results 4

In this chapter, the results from the analysis of the collected data about the case are presented. The data was obtained through interviews, the background material submitted from representatives from the project, and observations made in the premises where the project took place. The results have been presented in a more focused form based on the analysis, and categorized in a manner that was considered relevant for subsequent discussion.

The chapter is divided into two main sections; "Overview of the Project" and "The Use and Tailoring of Agile Methods and Practices". The first main section provides a case description that will be relevant for a general understanding of the project context. The last main section first presents the findings on the approach to agile method tailoring and then the findings related to the agile method and underlying agile practices.

4.1 Overview of the Project

The studied project was an IT project that took place in Norway, and was ongoing when this research was conducted. The customer was an organization with public transport as a core business, and maintained different digital sales -and communication channels towards their end users. These channels consisted of various digital solutions used by end users, such as web pages and apps which could be used to buy tickets, find information, and so on. The combination of digital channels were referred to as *omnichannel,* which is a term and approach used in business for managing multiple channels. Two important digital channels were the "Web channel" and "App channel", which involved development, management and operations of associated digital solutions. The customer had invested much resources in management and further development of the solutions in these two important channels. The focus of this study was therefore directed at the work that took place within these.

Table 4.1 shows a list of the most prominent stakeholders in the project. The most important stakeholders for this study are the suppliers Alpha and Beta, the customer (client), and end users.

Stakeholder	Description
Customer	The client of the suppliers. Organization with public transport as a core business
Supplier Alpha	External main supplier in the project. Responsibility for development, as well as management and operations in all the digital channels on behalf of the customer
Supplier Beta	External subcontractor in the project. Responsibility coinciding with the main supplier, but main focus on development activities
End user	The travelers who use the public transport services. Buys tickets and interacts with the transport organization through different digital solutions
Other stakeholders	Other stakeholders such as the transportation organization's own IT department, and other providers which the transport organization purchased or used digital services from

Table 4.1.: Overview of key stakeholders in the project

Having responsibility for all the digital channels meant that the suppliers worked with a wide range of tasks. Project tasks included everything from managing old legacy systems, rewriting existing systems to newer technologies that were easier to manage, to innovation and development of new solutions in more or less unknown areas. This entailed that the suppliers had to deal with very frequent changes and new tasks in multiple channels. Historically, the digital channels and associated solutions and systems had been developed and managed separately by a number of other suppliers before this project took place. However, the transport organization had recently gotten a more customer-oriented and competitive focus, and wanted to be more flexible in what products and services they offered to the travelers in the future. These were important reasons for now wanting more unified management of the different digital channels under one project, and also why an agile approach was selected. Three of the developers that were interviewed had worked in one of the earlier projects that took place in the customer's channels, and had valuable perspectives on the difference between then and the way they worked in the current project.

What was previously separate management of the digital channels, had resulted in poor cohesion between existing IT systems, and what could be described as solutions that were unnecessarily hard to manage for technical and design reasons. At the time, the systems were also operated separately from development in the customer IT department and at some later point outsourced for operation abroad. In this studied project, the suppliers had in cooperation with the customer made a number of improvement measures. Much time had been spent improving existing systems to make them easier to manage and further develop in the future. This eventually led to the introduction of DevOps, where the developers from suppliers Alpha and Beta now handled both development and operation tasks. These were regarded as important measures for being able to handle changes and delivering new solutions more quickly and efficiently.

The suppliers Alpha and Beta were hired to assist the transport organization under a four-year framework agreement. When the interviews were conducted, the project had lasted for one and a half years. Characteristic of the contract was also that it was a per-hour contract and not a fixed-price contract, which was regarded to have many benefits. Due to the responsibility for all the digital channels, many tasks were conducted in parallel at any given time within or across channels. In this sense, it can be argued that the case could be defined as a program consisting of several ongoing projects, but in practice it was referred to and managed as one large project.

At the time of the interviews, the project consisted of a total of 14 people from the suppliers, including three business developers from the customer who interacted daily with the supplier teams. Over time, the project had grown in size in terms of the number of people involved, and the plan was to continue scaling up in the future due to a need for more work capacity. The suppliers were located in the customer's premises, which enabled active customer involvement in the project. When the interviews were conducted, the suppliers were organized into three main teams working within and to some extent across the various channels. The teams were however frequently reorganized to handle the constant flow of new tasks and responsibilities, and some individuals worked across teams to make better use of available resources. Although the suppliers were organized into separate teams, an important principle in the project was that they first and foremost considered themselves as one whole team with shared responsibility. The informants stressed this idea, which seemed to have many collaboration and resource utilization benefits in practice.

It was apparent that an informal and largely dynamic organizational structure had grown in the project. Formally in the agreement, the project organization was hierarchically divided into different roles with different responsibilities, which appeared slightly different than the roles shown in Table 4.2. For example, the project manager was formally listed as a program manager, one was a technical project manager, one was responsible for the designers, and one responsible for the advisors. Underneath there were developers, designers and advisors. However, the project manager stressed that this was not the way they worked in practice. The organizational structure of the project was described to have become relatively flat and rather dynamic. Little emphasis were put on the different roles and responsibilities, other than those described in Table 4.2. For that reason, the project manager argued that a good graphical representation of the organizational structure was difficult to reproduce. For example, there were informal team leads that emerged depending on what they were working on at any given time. Individuals would often take on such informal roles for a period of time in one team, while they did not inhabit the same lead role in another team. The dynamic and informal distribution of roles not only involved leadership roles but also other roles. For example, a designer who had much previous experience working with agile and lean took on what could be described as an informal facilitator role. In Table 4.2, an overview of the various roles in the project at the time of the interviews is presented.

Role	**Main responsibilities and tasks**
Business responsible	Responsible for the project on the customer side
Project manager	Project management, organization and facilitation of the teams, project method and method tailoring, customer advisor and handling external relations, organizational and cultural development in collaboration with the customer, focusing on continuous optimization of development and operations
Interaction designers	Typical interaction design activities, such as defining the behavior of the solutions in which the end user interact. Worked closely with the designers
Designers	Worked mostly with graphical design tasks, defining how solutions should look visually. Worked closely with the interaction designers
Developers	Development activities and other tasks associated with the management and operation of the IT systems. Some developers had special skills and worked more on certain areas than others
Business developers	Internals from the customer who worked with the supplier teams on a daily basis. Responsibilities included giving input on how tasks should be prioritized, give feedback to the teams on completed tasks, and participate in testing and verification. Some business developers were referred to as "product owners" within the various channels
Advisors	Representatives from the supplier Alpha who worked with business management consulting activities on behalf of the customer. The advisors did not work with the same technical tasks as the developers and designers, however they were considered part of the team

Table 4.2.: Overall role distribution in the project

Figure 4.1 gives an impression on how the project was organized on an overall level in relation to the customer's business at the time when the interviews were conducted.

Although formally separate from the customer organization, the project were largely embedded as part of the customer's daily business activities. The *omnichannel forum* took place as a weekly meeting, which consisted of key people from supplier Alpha and the customer. This could be regarded as a steering committee that made higher level decisions for the digital channels.

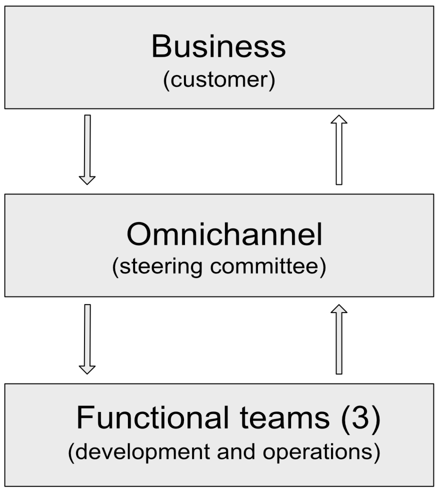


Figure 4.1.: Overall project organization

The project could in several dimensions be considered successful up to the current point. All the informants gave the impression that the agile approach worked very well, and had only improved since the start of the project. Two informants who had much experience with agile projects in the past even stated it to be the best project they had ever worked.

4.2 The Use and Tailoring of Agile Methods and Practices

In the following sections, the main results of the study is presented. The first section focuses on the results on the approach to agile method tailoring in the project. After this, a section is dedicated to results on agile practices that were deemed significant for enabling a high level of agility and continuous software delivery in the project. This includes how and why these were used as well as how tailoring took place for practices.

4.2.1 Approach to Agile Method Tailoring

The following results contains the key findings related to the agile method tailoring approach. It should be noted that the results across each section are largely coherent and somewhat overlapping, but that the headings indicates the core findings within each section. Within the sections there are also some information about the agile method and other important details which is not only related to the tailoring approach.

Flexibility in the Combination of Agile Practices

The project method was composed of several agile practices. Several of these are found in similar formats in various agile methods. The fact that the method was combined this way made it distinctive to the project, and not directly classifiable as a specific agile method. In an attempt to classify the method in the project, the informants were asked about which agile method it was most related to. The consensus seemed to be that it was closest to Kanban, but it was stressed that they they did not strictly confine themselves to nor define it as a particular method. Some different answers were nevertheless given to this question depending on who was asked. A developer believed that the method could be described as using a Kanban approach, but also believed that DevOps was descriptive of the method. However, others believed that DevOps was an important part of it, but not necessarily that it was descriptive of the overall method. This might have been because DevOps was very related to important software development and operation practices, but not as much for design work or other key project activities. The development model in the project could best be described as flow-based, where tasks to be done were visualized and managed in Trello[1] boards, which is a web-based tool for creating digital Kanban boards. The method thus had few similarities with Scrum, which uses an iterative and time-based (i.e. sprint-based) development model. Scrum also contains many roles, artifacts and activities that were not used in the project. It was apparent that some practices found in Extreme Programming (XP) were used in the project. An interesting observation was that, despite the method having similarities with methods such as Kanban and XP these did not seem to represent a boundary or framework for the tailored method. The fact that the method was agile was the most important: "*We do not talk about [which method we use]... We only use the common notion that we are agile*" (project manager). Overall, it was clear that the suppliers were not bounded by specific agile methods and was very flexible in that manner. An interaction designer suggested it may be difficult to name the method as a result.

[1]https://trello.com/

Tailoring Based on Past Experiences

It was apparent that the method was generally only classified as agile, and that they used various agile practices according to their needs. This can be illustrated in the following statement: "*...I perceive that you want a description of the method we have. While in my reality, I have a toolbox which I use as needed*" (interaction designer). Several used the term "toolbox" to describe this approach. In practice, this meant method tailoring was done by adding, changing, and removing practices throughout the course of the project. It appeared that this was mainly experience-based: "*What do we really need?... When you have worked in many different projects, you inspect what you did well there. What you need depends on the project. So, you are completely dependent on adopting agile methods that way; a toolbox where you can use what you need*" (developer). Several statements indicated that the tailored method was based on experience, and that they deliberately did not choose one or more known agile methods to stick to by default. A developer who had worked with more specific agile methods in the past did not experience the lack of a clear method framework as something negative in this project: "*I have not really experienced that there is anything negative in not standing by a specific method*" (developer). As was mentioned, the general consensus was that the method and way of working in the project worked very well at the time of the interviews.

Focus on Minimum Overhead and Formalities

The project manager thought a good description was that they worked according to agile principles: "*I like to think we are working according to agile principles, I think that is a good description. And then this also includes reducing scope, and making sure we get rid of the technical barriers that prevent us from working in an agile manner. When you resolve these challenges, it somewhat comes into place by itself*" (project manager). This brings us to one important principle they worked towards, which was to maintain "*as little overhead as possible in everything they do*" (project manager). This idea also applied to the method. They wanted a method that included only the necessary, and which they experienced to give value to the project. In this context, several informants pointed out that the method consisted of few formal mechanisms and rules, which was deliberate so that they could be more adaptive and make adjustments to the way they worked as needed. The project manager thought this was an important factor in the project: "*What I think works well is that we constantly make changes depending on what we are working on. I think we are quite adaptable*" (project manager). It was however considered necessary for the project manager to give the teams some guidelines, and especially since the number of people involved in the project increased. However, to strike a balance was still important: "*... that is the key. To constantly look for a balance between how little of this 'overhead' we can do without, and what we need to introduce to maintain control. We must always make sure we are on that limit, and that we do not introduce mechanisms that are unclear and unnecessary*" (Project Manager).

An interesting observation from the analysis was that the informants emphasized that there were few "rules" in the method, which also reflected what the project manager said about reducing overhead. A Scrum approach had been used in a previous development project in one of the digital channels, where three of the interviewed developers had worked. A developer said that "*in [the method of a previous project] compared to now we have very few rules, few definitions*" (developer). Similar statements also appeared in interviews with other informants. Two developers indicated that little regulation and formalities also meant that they could spend more time on actual product development, and that the speed of development had increased considerably over the previous project. Although formal regulations and definitions in the method were few, the teams did adhere to a number of practices to guide development and design in the project. The practices that were arguably most predominant in the overall method and used by all of the teams, including both developers and designers, were:

- Kanban boards
- Daily stand-up meetings (per team)
- Common stand-up meetings (weekly stand-up common for all teams)
- Self-organizing and cross-functional teams

These and other practices will be presented in Section 4.2.2. Through observations and analysis of the project it was apparent that the teams were largely flexible in how they used practices. Stricter regulation seemed to instead be found in the more technical development practices, and not so much in management practices or the overall method. An interaction designer referred to these as "micropractices": "*... [the way we work] is less square, but it is very regulated. But the regulations are in how to check-in code, how to write things, how to build things... and not so strictly in how we work in the room or talk to each other. So it is more for 'micropractices'... there's a lot of regulation, but not so many rituals and such*" (interaction designer).

Continuous and Informal Tailoring

Product development and innovation took place continuously within and across the channels. It was evident that the teams were driven by a goal of creating as much value for customers and end users as possible. The findings suggest that tailoring was required to maintain good productivity when dealing with many different tasks of different character and size. In this context, the project manager and an interaction designer stated that it was important not to get stuck in old habits, but rather to focus on continuous evaluation and improvement also in methodological aspects. Thus, a notable finding was that method tailoring was a continuous and conscious activity in the project. However, it appeared it was mostly done in an informal manner based on emerging challenges or needs, and not as a formal and structured process or activity. A designer put forward that "*the method evolves when things do not work*" (designer). From the analysis of the interviews it seemed evident that the teams had a mentality for open dialogue and will to improve in the project environment. It could seem that this decreased the need for other formal measures like retrospective meetings: "*It is easier to just talk about it at once and fix it, during stand-ups and ongoing... If there is something that slow us down, we try to fix it at once. But of course, this require an environment where this works. If the environment was not suitable, then it would likely require us to have a retrospect meeting occasionally*" (developer). Such a culture in the project environment was something the project manager figured was important to promote: "*...wherever possible, I would rather promote open dialogue and build on the cultural aspects so that people actually dare to bring up things along the way; 'no, we should not do that, we have to do it like this'*" (project manager).

Purposeful Tailoring with Frequent Feedback from Teams

The informal form of method tailoring took place through discussions between teams and management, either within meetings or in the open landscape office. Implementation of major improvement measures and tailoring of the overall method was described as being first and foremost the project manager's responsibility, but was often based on suggestions that came from others in the teams: "*It is the project manager that [tailoring] depends on the most... But the wishes or the changes do not come just from him. Suggestions are often done by us, and then he implements it*" (developer). Several informants pointed out that there were some particularly committed and experienced individuals in the project who were more actively involved in discussions regarding the method and which adjustments should be made. The project manager argued the same: "*...those who are committed and who cares about it are usually those who are involved in those discussions*" (project manager). The informants told that the overall competence and commitment of the people involved in the project was very high, and that many had much previous experience with working in agile environments. On occasion, suggestions had been made by individuals to the project manager based on their own experience and knowledge which resulted in changes in the method which other informants experienced as improvements. An experienced interaction designer were among those who were engaged in method improvement discussions on a regular basis: "*It is not like everything we do is cyclical and we work on the same things over and over... A lot of what we do here therefore involves continuously tailoring how we are organized and what we do, to the tasks we are doing. There is a lot of meta-talk between me and the project manager to reflect around how we are doing things, and what challenges we have with good or bad habits*" (interaction designer). For the project manager, an important responsibility was to continuously evaluate and optimize the agile approach in the project by reducing overhead. The project manager emphasized that an important and related task was then to develop the right organizational culture and create the right prerequisites for an agile environment in collaboration with the customer.

Ad hoc Tailoring on the Team Level

While the project manager implemented the more major changes to the overall method, there existed much freedom and flexibility in how the teams worked. The teams were relatively free to choose how they used some of the agile practices. For example, the teams made many individual adjustments of their use of Kanban boards. Thus, ongoing ad hoc tailoring was done on the team level without the project manager being directly involved. As mentioned, it was however important that some guidelines were given which the teams followed. This was necessary to ensure some overall governance: "*...there is a high degree of freedom in how to do things in different teams, but it is important that we have some similarities*" (project manager). For example, although there were some flexibility in using Kanban boards, some directions were given by the project manager on how the boards should be set up. As was mentioned, all teams and all team members committed to the use of the four predominant practices Kanban boards, daily stand-ups, common stand-ups, an self-organizing and cross-functional teams. Thus there were important similarities in the method across teams. More on the use and tailoring of practices is presented in the following section.

4.2.2 Agile Practices and Tailoring of Practices

From the data collection and analysis it was found that the method was composed of a variety of agile practices. In the following section, I will focus on the prominent practices used as part of the agile method. Practices may here be regarded as activities, principles, techniques, tools, and the like, that the teams actively and systematically used when executing their work.

A form containing a list of practices was used as a basis for revealing what practices were used in the project. The survey was done during the interviews, and was based on the list of agile practices from a reliable agile and lean usage survey (Rodríguez, Markkula, et al., 2012). The same list of practices was presented in the theory chapter (Section 2.2.4). The form used during the interviews is found in Appendix A.2. A selection of informants participated in the survey during the interviews and explained their choices. Not all informants participated in this as the purpose was not quantitative data collection, but to get an initial overview of practices as a starting point for further qualitative data collection. More in-depth questions regarding how the practices were used and tailored was asked during the interviews. It should be noted that the list of practices included in the survey is not a complete list of existing agile practices. Therefore, qualitative data collection was necessary to identify whether other prominent agile practices were also used in the project.

A notable finding was that many of the agile practices from the survey were used in the project. 11 out of 16 practices were extensively used in the project according to the informants, while 5 of them were used very little or not used. Although many of the practices from the survey were used, these were often part of a more comprehensive practice and not necessarily recognized by the same name in the project. An overview of the practices in the survey and the corresponding practices in the project are presented in Table 4.3. This also includes some of the additional and related practices that were prominent, but not covered by the survey.

Practice in survey	**Practice in project**
Prioritized work list	There was extensive use of Kanban boards in the project. Contained backlogs where tasks were prioritized and other columns to visualize status on individual tasks. Primary tool for visualizing and managing work in the project

Practice in survey	Practice in project
Daily stand-up meetings	Each team had separate stand-up meetings every day before lunch. However, weekly stand-ups common for all teams were also conducted (i.e. common stand-ups)
Active customer participation	The supplier teams were located in the customer's premises. Business developers from customer were involved daily in the development. Customer representatives and other stakeholders were also involved on a daily and weekly basis through a number of other meetings and arenas
Frequent and incremental delivery of working software	Changes and new software features were deployed several times a day to production and testing environments in an incremental fashion. Thus, *continuous and incremental delivery of working software* is arguably more descriptive for this project
Automated builds	The teams had developed and configured a system that allowed for automated building of code into working and deployable software. This was part of the DevOps and Continuous Deployment practice in the project, which made it possible to deliver changes and new features quickly to production
Continuous integration	This is a given, since it is a prerequisite of the Continuous Deployment practice in the project (see Automated Builds). Code integration processes were automated and performed more or less continuously by the developers
Refactoring	There was a continuous focus on improving existing applications in the project to make them better and easier to manage and operate. This meant that the developers spent much time rewriting and improving code, but also system architecture, underlying services etc.
Collective code ownership	An important principle was that although the suppliers operated as multiple teams working on different tasks, they viewed themselves as one whole team with common goals and responsibilities. Shared responsibility also applied to development, where developers had a shared responsibility for maintaining and developing the software systems

Practice in survey	Practice in project
Self-organizing teams	The teams in the project were largely self-organizing. Developers and designers made many decisions on their own. They were also to a large extent cross-functional. Each team was organized so that they possessed the necessary skills to execute their tasks
Unit testing	The developers did quite a bit of unit testing. Although unit testing specifically seemed to not be done as extensively as some of the other practices. Testing in general was an ongoing activity in various test environments which the developers could set up quickly and on demand
Pair programming	The developers frequently worked together in pairs on the same code. This was also observed during my visits to the premises. However, this was not done all the time, and programming was primarily done individually

Table 4.3.: Overview of identified agile practices from survey and additional practices (Survey in Appendix A.2, based on Rodríguez, Markkula, et al. (2012))

It is apparent from Table 4.3 that the list of practices in the original survey were not the only practices used in the project. For example, the teams used weekly common stand-up meetings in addition to daily stand-up meetings. Several of the practices were also part of a more comprehensive practice involving other practices. For example, priority work list was only part of the use of Kanban boards. The teams were not only self-organizing, but also largely cross-functional. Some practices also overlapped, such as active customer participation which involved a number of arenas such as daily stand-up meetings.

An interesting finding from the analysis and observations was that practices were also tailored to the project context. The needs of the teams continued to change throughout the project based on changes in tasks and situation. This sometimes resulted in adding, changing, and removal of practices.

In the following sections, the results on the most prominent practices in the project from Table 4.3 and the analysis of the qualitative data is presented. The following headings were chosen to be appropriate: *Kanban Practices*, *Self-Organizing and Cross-*

Functional Teams, *Stand-up Meetings*, *Active Customer Participation*, *Continuous and Incremental Delivery of Working Software*, *DevOps Practices*, and *Other Practices*.

Kanban Practices

The perhaps most fully adopted agile method in the project was Kanban. This involved extensive use of the web-based tool Trello[2], which can be used to create digital Kanban-like boards. The boards were mainly used to visualize work and manage the flow of individual tasks through different phases, limit work-in-progress (WIP), and measure flow using a cumulative flow diagram (CFD). Despite being clearly inspired by Kanban, they did not formally confine themselves to Kanban's rules: "... *we have not implemented it as a Kanban project, because it is not. Like, deliberately. Because when adopting Kanban as a method, you have three rules you commit to*" (developer). The Kanban boards were portrayed as the centerpiece of the agile method: "*It is very central. In a way, everything revolves around [the Kanban boards]. It is where you get all status, all flow, and what all the planning is centered around. The tasks to be done and when to do them*" (developer). All the teams in the project used Kanban boards on a daily basis. All the tasks in the project were placed in boards to give everyone a visual overview of work and status: "*I believe everyone is comfortable that it is the way one can visualize what people are working on and the status of tasks, and when it should be finished*" (developer). The suppliers had also involved the customer in the use of the digital Kanban boards. Thus the business developers could add, modify, and follow progress on tasks. Some business developers were referred to as the product owners, responsible for backlogs in their respective channels. However, both business developers and the teams added, modified, prioritized, and verified tasks in the boards.

The teams created multiple boards for different solutions and purposes. For example, in the Web channel they had one board for management and operations tasks and another board for a new solution they were working on. The division into multiple boards also applied to the other channels. For very large tasks, they were broken down into many smaller tasks and assigned a respective board. An important principle was to break down tasks into as small work packages as possible. This was important in order to reduce the scope of what was being worked on at any given time. Breaking down work into small and manageable tasks made development less resource demanding in general, and tasks could be completed, tested and delivered faster. According to the project manager, the breakdown of tasks led to a normalization of task sizes into small and somewhat larger tasks in practice. This again led to a stable burnrate in number

[2]https://trello.com/

of tasks completed, which they considered to be accurate enough to base estimates on. However, the teams did not do estimation of individual tasks, but informally reported rough estimations on when they expected overall solutions to be finished. This was perceived to save time and increase productivity in general, and was possible due to a high level of trust between the teams and the customer.

A board contained several columns where short descriptions of tasks were placed in the form of digital "cards". The project manager gave the teams some guidelines for how the boards should be set up so that data could be exported from the boards for measuring flow and progress. Beyond these guidelines, the boards could contain various columns and be configured in different ways. Progress was measured in the number of tasks flowing through the different columns and visualized using a cumulative flow diagram (Figure 4.2). This was a useful way for the project manager to keep track of the overall flow in development, and whether there existed any bottlenecks in the development process that required further investigation or measures. The following columns were typically found in the boards:

- **Backlog:** Contained the tasks to be done. Tasks were identified and placed here by the customer, but also by the supplier teams. The teams typically broke down the tasks into smaller and more manageable tasks.
- **Ongoing:** When a task was assigned to someone and started, it was typically moved to the "Ongoing"-column. The teams had decided on limits for how many tasks that should be ongoing at once, thus limiting work-in-progress (WIP). For example, one board had a limit of five simultaneous tasks.
- **Test:** Completed tasks were typically placed in a "test"-column. This implied that the task had been implemented in terms of code and/or design, and were ready to be tested.
- **Ready for release:** Tasks were moved to "ready for release" after testing. Most tasks awaited verification by the business developers before they were deployed. However, many tasks were also deployed without direct verification from business developers if the developers were confident that the tasks were ready.
- **Done:** When tasks were verified, or considered by developers to be ready, they were typically deployed to production and placed in a column for completed tasks.

An interesting finding was that the teams frequently made adjustments to the Kanban boards. The boards had been through several iterations on how they were used and configured. An interaction designer was one who pointed this out: "*Trello has been*

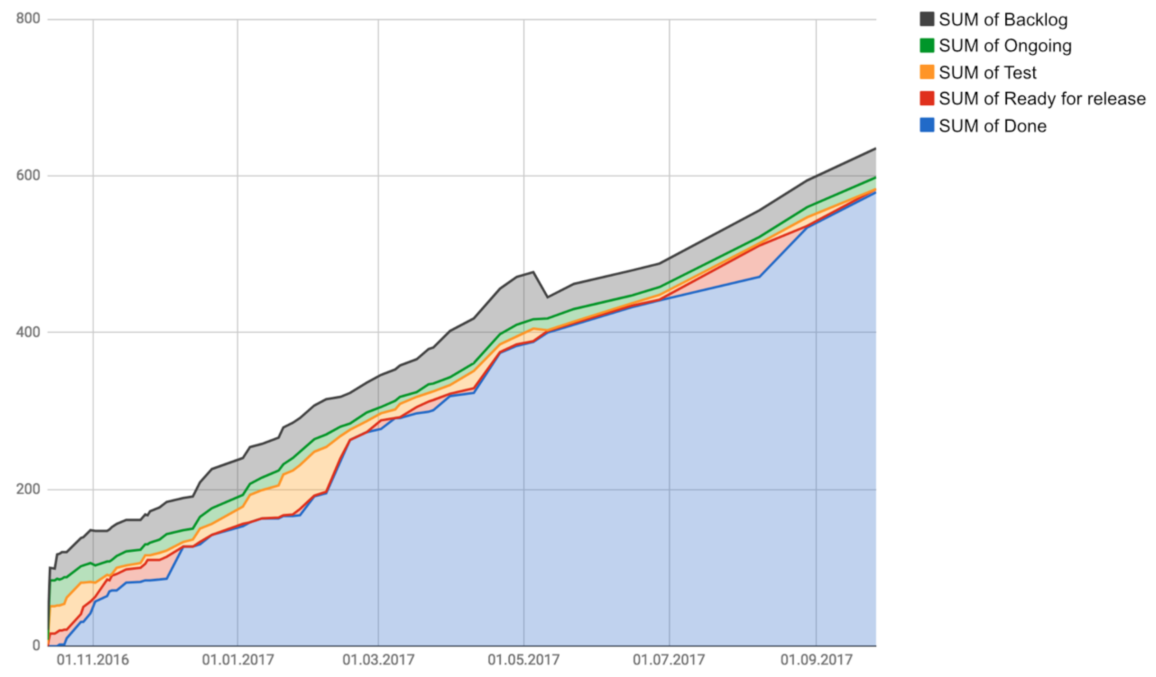


Figure 4.2.: Cumulative flow diagram from the project

through many iterations in how we use it. With regular reflection about what we should do... There is a lot of discussion about how many and what columns we should have, and what should be on the cards" (interaction designer). For example, a board for management of existing solutions could look quite different than a board for a new solution and with regular changes. New solutions with dedicated boards typically had a "MVP"-column (minimum viable product), in addition to a "backlog" column, where high priority tasks were placed. For new solutions the goal was to prioritize and complete the most important tasks in order to have a working MVP deployed to production as quickly as possible. The idea was that a 70% finished solution was worth more in production where they could further iterate and get feedback from stakeholders, rather than attempting to create a 100% solution in a test environment. The project manager pointed out that adjustments of the boards were an ongoing activity: "... *the nature of the tasks we are working on right now are large in scope, or difficult and unmanageable, which you never really completely finish... this always starts a discussion whether we have the right configuration in our boards. And then we have made many adjustments. Some adjustments exist for a while and then we go back to a previous configuration. But, of course, there are those who are committed and care about these things who are involved in these discussions*" (project manager). The nature of the tasks had a major impact on how the boards were set up. Changes in circumstances in the project also led to adjustments of the boards. For example, if a product owner (i.e. business developer) in a channel was unavailable and unable to verify completed tasks or answer questions, one team had set up a separate "parking" column for the tasks that were unclear. The teams generally challenged the tasks that were added to the boards if they were in doubt about the value or usefulness of the feature: "*...if we think this gives little value for the user, we move it to the "parking" column. Normally, we [the team and business developer] will discuss these things during stand-ups. But we found that if they are unavailable and cannot participate, then we have to put it somewhere in the meantime*" (developer). As one informant mentioned, this seemed to contribute positively to the overall flow and overview of tasks in the board. The fact that the Kanban boards were digital made it easy to reconfigure the boards to changing needs. Figure 4.3 illustrates a typical Kanban board layout from the project, and how teams could configure the boards by adding additional columns in between the baseline columns for improved task flow.

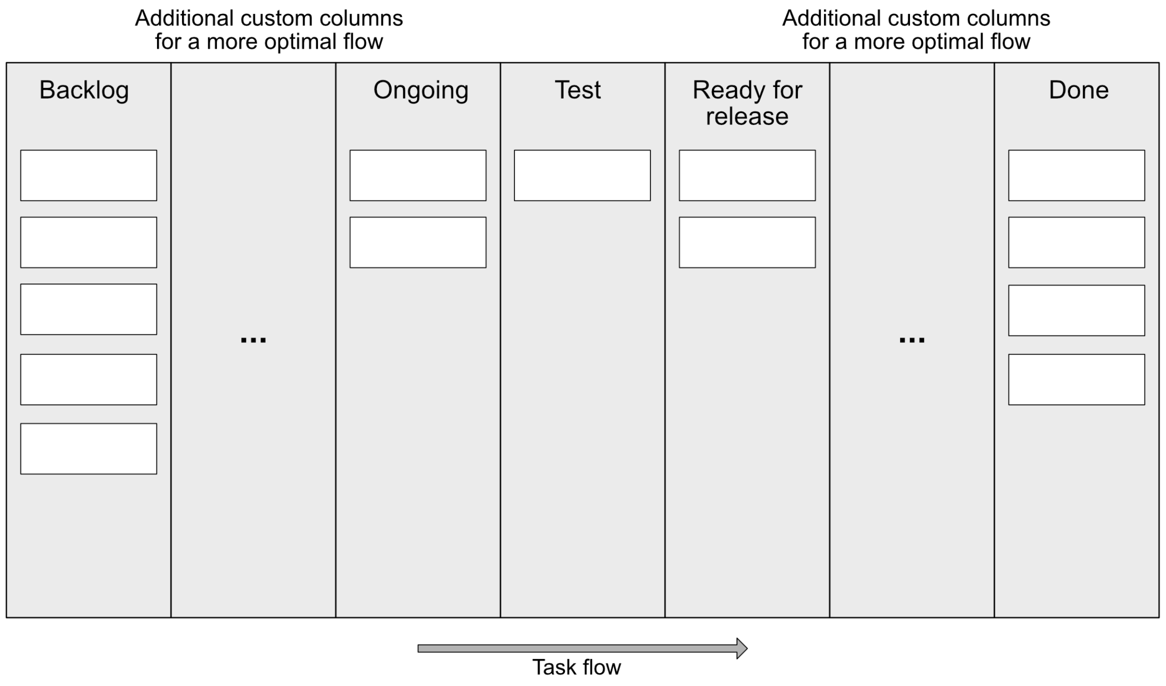

Additional custom columns for a more optimal flow
Additional custom columns for a more optimal flow
Backlog
...
Ongoing
Test
Ready for release
...
Done
Task flow

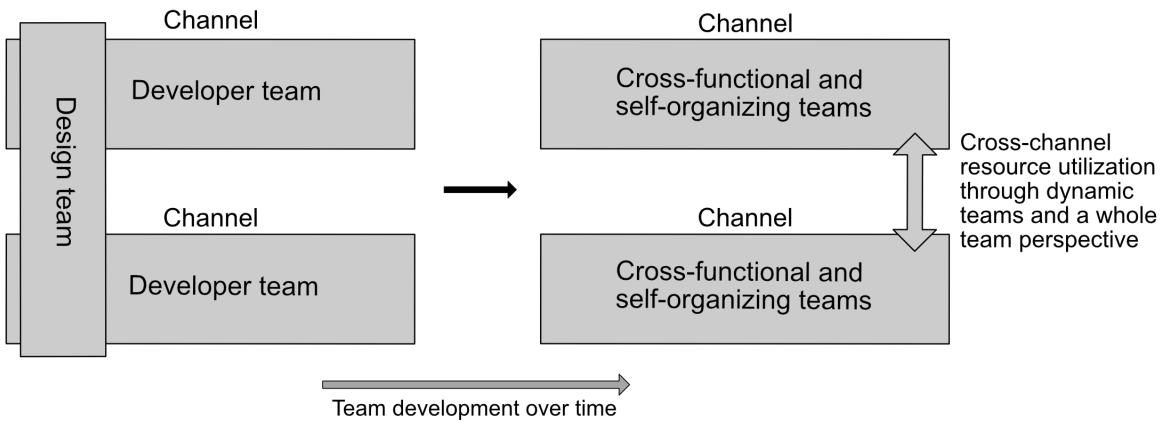
Figure 4.3.: Sample configurable Kanban board layout from the project

Self-Organizing and Cross-Functional Teams

During the interviews and observations, participants stressed that although they were three operative teams at the time, a principle of being one whole team was important. Statements suggested that this was important to prevent the teams from getting rooted in the channels, and enable utilization of resources across areas. A developer argued it was important "*to have people in the team as a whole who can work in many areas, where they are needed the most. This creates more flexibility*" (developer). The project manager argued that the organization of teams could best be described as "*...one team of internal dynamic groups*" (project manager). In practice, the teams could be reorganized as needed to utilize resources and create self-organizing teams: "*We are divided into different teams, which are not constrained boundaries on our part. We want as little overhead as possible, so it it very dynamic. We establish a team that works on something for a period, and then break it down again. We try to achieve what you elsewhere define as fully independent autonomous teams*" (project manager). The teams were also to some extent cross-functional and consisting of the skills needed to execute their work independently. However, the teams seemed to neither be fully self-organizing nor fully cross-functional at the time of the interviews. Creating more self-organizing and cross-functional teams was an ongoing and important improvement process according to informants.

One of the informants described the teams as semi-autonomous teams. At the time of the interviews, the teams managed most of their daily work and followed up on tasks on their own. A developer argued that the consequence of this was that the teams had "*much freedom, but also great responsibility*" (developer). In order for the teams to become more self-organizing, several argued it was necessary for the teams to obtain more decision-making authority. The project manager argued that this was necessary to improve in the future: "*[if tasks are decided at a high-level] and arrive at team level, the [tasks] are regarded a 'feature' and you lose understanding of why. At the same time, you lose the feeling of ownership that is necessary in order for the team to actually innovate and improve the solutions continuously. So for our part, it is very important to move that responsibility down to the teams*" (Project Manager). This was an ongoing process in close collaboration with the customer. The informants argued that the teams were more self-organizing and cross-functional than before, but that more improvement work remained. The project manager stated that this involved a large amount of organizational and cultural development: "*...we have come to a point where we can say we have more functional teams, but it is an ongoing process right now. There is a lot of organizational and cultural development involved in this*" (project manager).

Another notable finding was that the teams had evolved from being organized mostly based on roles, to becoming more cross-functional and operate more across areas. Earlier in the project, all interaction designers and graphic designers were organized into one team separate from the developers. At the time, the design team worked across the digital channels, while developers worked as teams in separate channels. A designer argued this to be nonproductive. Design decisions had to be made by everyone in the design team, which consequently led to time-consuming discussions to reach consensus. This led to a splitting of the design team and merging with the development teams. Thus, the teams became more cross-functional and both design and development decisions could be made separately within each team for the solutions they were responsible for. The designer experienced that, as a result, decisions were made much faster, which in turn led to more agility: "*Now we manage the channels separately. In order to be a little more agile, really. Now I have an area of responsibility where I can make decisions which makes the process faster. Instead of having to address all issues in plenary and reach consensus all the time*" (designer). Figure 4.4 illustrates the overall development towards more self-organizing and cross-functional functional teams based on the analysis of the collected data.



Design team
Channel
Developer team
Channel
Developer team
Channel
Cross-functional and self-organizing teams
Channel
Cross-functional and self-organizing teams
Cross-channel resource utilization through dynamic teams and a whole team perspective
Team development over time

Figure 4.4.: The over time development towards more self-organizing and cross-functional teams

The teams occupied people with much previous experience working in agile environments. Informants perceived the competence level of both design and development personnel to be above average in the project. The informants argued that the responsibility and trust they were given by management and the customer led to a sense of ownership of the work and that people were very committed. These were pointed out as important factors for autonomy in the teams. For example, the project manager said that he did not "*...need to follow up on each individual because they are so self-driven and skillful. I have experienced projects with a different competence mix... And then I have had to introduce more defined mechanisms and arrangements for far smaller teams than this in order to utilize capacity*" (project manager).

Stand-up Meetings

All the teams participated in both daily and weekly stand-up meetings. The two arenas were used for different purposes, and were conducted in somewhat different formats. In addition to data gathered through interviews, these arenas were observed during my time with the case. In total, two daily stand-up meetings and two common stand-up meetings were observed.

Daily stand-up meetings were considered a very important arena for coordination within the teams. A notable observation was that collocation in the customer's premises enabled business developers to participate in the daily stand-up meetings. Several informants pointed out that daily stand-ups were very important for adequate customer involvement. This was particularly because the business developers and other customer representatives were located in another part of the premises. This separation made scheduled meetings still a requirement, despite a short distance to the customer representatives: "*...we end up being a bit separated. And, as a result, we resort to stand-ups and other meeting activities with the customer*" (developer). The meetings were held for each team in small rooms and the meeting took place while standing. This was done deliberately to keep the meetings short. One observation I made was that the daily stand-up meetings were very centered around the Kanban boards. During the meetings, the team discussed the various tasks and their status, assigned people to tasks and whether they should be moved to other columns. They also made quick changes in both the board configuration and in tasks for multiple boards. The tasks that were unclear could be addressed with the team and business developer: "*We mostly focus on the boards and discuss the tasks we need feedback on, or need to have explained a little more*" (developer). Daily stand-ups had been through several adaptations: "*...having stand-up is really useful. But how much stand-up should we have? You can not have stand-up meetings for everyone all the time, and thus we have tried different approaches*" (interaction designer). An interesting remark was that stand-ups were also referred to by informants as an arena where discussions on potential improvement measures for the method took place: "*...changes [to the method] are also proposed both during stand-up as well as ad hoc afterwards*" (project manager). The daily stand-ups were supposed to be timeboxed to 15 minutes and facilitated by a representative from each team. However, the meetings seemed to be in held in a relatively open format. Discussions about many topics and issues, such as the ones described above, could therefore be brought up. This, and the fact that the customer was involved, was perceived to often make the daily stand-ups longer than intended.

Once a week, the suppliers held what they called a common stand-up meeting. This meeting involved all persons from the suppliers including advisors, thus it was quite crowded. The customer was however not present. The purpose of this meeting was to synchronize as a whole team. I learned that the common stand-ups had been introduced later on in the project based on an emerging need to be able to stay more synchronized and coordinated. This was in multiple cases mentioned as a case of method tailoring, in which the teams had assessed which practices were useful and what they needed: "... *we do [practice assessments] all the time; 'does this give any value, do we really need it?' The common stand-up is something that was introduced later, because people wanted a more complete picture*" (developer). Another developer described the introduction of common stand-ups as a way to stay more focused on the principle of being one whole delivery team with common responsibilities: "*We started off with only normal stand-ups... but then we recognized a need to become one whole delivery team, across apps and advisory and so on. And as a result, the common stand-up was introduced. We have an idea and a wish that everyone can deliver on everything. And that is why we use common stand-ups, so we can share knowledge and insight*" (developer). Common stand-up meetings were formally implemented by the project manager. The project manager commented that in the near future when they planned to scale up with more people and teams, they would likely need to change the format of the meeting to make it more manageable. He further stated that such assessments of potential method tailoring was something that he continuously considered.

Figure 4.5 summarizes stand-up meetings as having many important functions in the project. More on the role of stand-up meetings for customer involvement is presented in the next section.

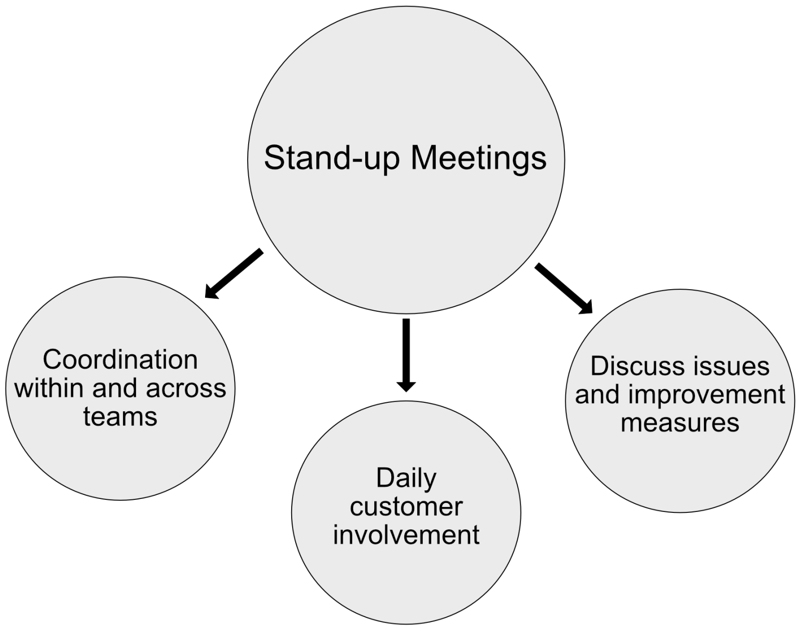

Stand-up Meetings
Coordination within and across teams
Daily customer involvement
Discuss issues and improvement measures

Figure 4.5.: Important functions of stand-up meetings in the project

Active Customer Participation

The customer was involved on a daily basis in the project through several arenas, including daily stand-up meetings. Several informants argued that collocation was essential since the project involved continuous development and innovation of all of the customer's digital channels and corresponding products and services. Thus, the project was to a large extent driven by a constant stream of changing requirements and new tasks. The high rate of change and high degree of innovation in the project made it very important to work closely with the customer in order to receive frequent feedback and adhere to the customer's goals: "...*we always want feedback on what we are doing, and whether we are working in the right direction*" (developer). A developer argued that "*there is nothing that beats oral dialogue... So we are completely dependent on staying here and work closely with the customer to understand the customer and know what they actually need and what their goals are*" (developer). However, the distance between the teams and the customer representatives created a barrier to continuous oral dialogue with the customer. This could have been improved by having the business developers sit closer and work as a part of the teams: "... *[the business developers should] sit together with us, optimally. So they know what is happening all the time, and prioritize tasks together with us to ensure that we always work on the most important tasks*" (developer). During my observations, the business developers occasionally came along and discussed tasks and gave directions to team members also outside of meetings.

To ensure adequate customer involvement and compensate for distance, multiple arenas had been introduced. An interaction designer described that: "...*we have a need to stay updated on what is happening. And the first thing we [the development teams] do is to sit together. One challenge is that even though the customers sit down the hallway, they are still far away. It would have helped if they were five meters away. But that is why we introduce other measures, such as stand-ups, to make sure they are in sync with the development teams*" (interaction designer). Many coordination arenas were identified during the interviews, and the customer was involved in many of these. Without going into particular detail, among arenas that involved the customer were:

- Daily stand-up meetings
- Omnichannel forum (weekly steering committee meeting)
- Ad hoc meetings

- Online chat using Slack[3]

Several informants stated that they experienced a very high level of trust between the customer and the teams in general, which they believed to have emerged partly due to the team's ability to deliver good and consistent results. The informants believed that the trust in turn lowered the need for formalization in the involvement of the customer and in the method in general. This in turn seemed to contribute positively to agility and productivity according to some statements of the informants.

Continuous and Incremental Delivery of Working Software

Frequent and incremental delivery of working software is a fundamental principle for agile software development in general. In this project, working software was delivered in a continuous and incremental fashion, and thus more frequent than in many agile contexts. Note that this is not a single independent practice such as stand-ups, but a fundamental practice which depended on the overall combination of agile practices in the project. This section presents some general findings regarding what continuous software delivery entailed in the project.

Most tasks were identified, developed, tested and deployed to production consecutively, which in practice meant that changes and new functionality were quickly available to end users as well as to customer representatives. As previously mentioned, new solutions entailed development of a working MVP (minimum viable product) before it was deployed to production where iterations continued in terms of small increments. Everything that was deployed to production was not necessarily made available to all end users right away, but to segments of users or only for internals for them to test the solutions. A developer exemplified this for a new solution they worked on in the Web channel: "*...we implement the [solution] incrementally and deploy it to production along the way, but we do not activate the services and so on. They will not to be visible to most users, but we develop it and deploy it all the way to production*" (developer). Over time, the deployment rate had increased in the project until the point where working software was deployed several times a day to the different solutions and systems in the digital channels: "*...we [deploy] continuously, multiple times a day, every day*" (project manager). One of the advantages of delivering continuously to production was that feedback from both the customer and end users could also be obtained continuously through metrics in the systems, through customer service, daily customer involvement, and so on. This created opportunities for the teams to handle changes and make improvements to the systems on a continuous

[3]https://slack.com/

basis, and deliver these quickly back into production: "*It is all about getting the first version released into production as fast as possible, because that is more important than delivering a full-fledged product. Then you can get feedback quickly on whether you have made the right choices. And if not, we can make corrections much more quickly than before... You no longer receive surprises like you did in the old days when everything was delivered and you thought you were done*" (developer).

A couple of technical agile practices were essential to enable quick delivery of software features into production. This included automation of code integration, builds, and deployment through the practices Continuous Integration (CI) and Continuous Deployment. These could be understood to be part of the DevOps practices in the project, which is further presented in the following section.

DevOps Practices

One developer described the method in the project as "agile with DevOps". We know from research (Section 2.2.3) that there is seemingly still no agreed upon definition of DevOps to date. In this project they seemed to regard DevOps as a set of practices and an important part of their agile method. It was also regarded a prerequisite for continuous software delivery. Thus, very much in line with views on DevOps in current research.

Some time after project start, a deal was made with the customer that the teams would take over the responsibility of what was previously external operation of all systems. DevOps practices were introduced as the supplier teams gradually moved existing solutions to a cloud-based platform. The chosen cloud service was Amazon Web Services (AWS), where they used a Platform as a Service (PaaS)[4] solution. The choice of PaaS removed the need for the teams to manage the underlying infrastructure like hardware and operating systems. As a result, the developers could handle deployment and operation tasks on their own as a sideline activity, while still devoting most of their time and resources on applications development. There were no longer a distinction between developers and operations personnel, hence DevOps: "*There is no operations personnel, no one occupy that role... It is DevOps*" (project manager). An important opportunity made with DevOps, and the associated merger of development and operation, was that the developers could introduce a Continuous Deployment practice by automating the processes all the way from development to execution. This ultimately made software delivery faster from a technical standpoint. Identified DevOps practices in the project were the integration of development and operations,

[4]https://aws.amazon.com/types-of-cloud-computing/

automation, monitoring, and measurements. Moreover, an interesting remark was that the development and operations process on an overall level seemed to be largely similar to the DevOps approach illustrated in Figure 2.5 in the theory chapter (Section 2.2.3).

It was argued that DevOps and related practices led to many other advantages, such as enabling the developers to better handle application problems. The project manager introduced the principle "you build it, you run it" in this context: "*...we put much emphasis on the idea 'you build it, you run it', which we believe is a very important principle. We are much better rigged at handling application problems than operations personnel that do not know the application at all*" (project manager). A developer believed that DevOps and Continuous Deployment made it possible to utilize the benefits of having very frequent feedback available through active customer involvement.

Other Practices

There was also extensive use of the practices pair programming, refactoring, and collective code ownership in the project. These practices are often associated with the agile method Extreme Programming (XP). The purpose of these are typically to improve code quality, as was described in the XP literature in Section 2.2.2. Pair programming was used when the developers found it useful and convenient, but not consistently at all times as recommended in XP. Moreover, the informants did not specifically refer to XP as a starting point for adopting these practices. It could seem that they were used on the basis of past experiences.

The teams had to some extent used retrospectives earlier, which is also found in Scrum (Section 2.2.2), but stopped using it over time. Whether they should continue using it or not was an ongoing discussion. As mentioned, the teams seemed to have an open dialogue and a mentality in which they solved problems in other available arenas. The project manager stated: "*Retrospectives is still a discussion sometimes... I really believe that retrospectives may be important. At the same time, I see a challenge using retrospectives for the entire team because we are so many. What we do now is that we have a very open dialogue in the stand-ups. So that is primarily where we bring up these things. And additionally, I spend much time walking around and talking to people*" (project manager). The project manager commented that he did consider implementing other approaches to retrospectives in the future, perhaps for each self-organizing team.

Overall, these findings again confirms that practices were both tailored and that practices from other agile methods than Kanban and DevOps were used.

5 Discussion

This chapter will focus on discussing the results against the problem statement with focus on the underlying research questions. This is done by interpreting and contextualizing the results by using the scientific knowledge base provided in Chapter 2. The last two respective sections of this chapter includes evaluations of the implications and the limitations of the research.

5.1 Problem Statement

The problem description in Section 1.2 presented the following problem statement to be addressed in this study:

> *How can agile methods be tailored with aim towards continuous software delivery in projects?*

Two more specific research questions were decided based on the problem statement to be further investigated in detail. The two following sections will present these, and discuss the results with regard to each of the research questions respectively.

5.1.1 Characteristics of the Agile Method Tailoring Approach

The first research question was given as follows:

> **RQ1:** What characterizes an agile method tailoring approach in a modern software development context?

With RQ1, I wanted to identify prominent characteristics of how method tailoring was performed in a modern software development project context. As a reminder, a modern project context here refers to a context in line with recent trends in agile software development where software delivery is done continuously, as in the case in this study, rather than on the typical two to four week basis. The following discussion will focus mostly on results on the tailoring approach found in Section 4.2.1, but the other results on agile practices are also included in the discussion when appropriate. The theory on method tailoring (Section 2.3) will be particularly relevant for this discussion.

In the results we learned that the suppliers had avoided confining to a pre-selected method framework when tailoring their agile method. Agile method tailoring was in this regard done by combining agile practices from various agile methods such as Kanban practices on one end, to stand-alone practices that may not have directly originated from an agile method such as common stand-ups on the other. Previous research on method tailoring in more traditional agile contexts presents cases where one or two methods were chosen as an initial framework for tailoring agile methods, such as combining practices from Scrum and Extreme Programming (XP) specifically (e.g., Conboy and Fitzgerald, 2010; Fitzgerald, Hartnett, et al., 2006). In this context this was not the case, which had the apparent benefit that they were much less restricted in which practices to use. Thus, this provided much flexibility and adaptability for the suppliers to combine and tailor agile practices throughout the project as they needed.

Furthermore, it is interesting to compare the results with method tailoring approaches in the literature that was supplied in the theory chapter (Section 2.3.2). Based on the fact that no specific method was pre-selected in the project, the method tailoring approach had very little in common with contingency factor approaches proposed in literature. Contingency factor approaches require much experience with many methods and proposes that one is selected that best suits the given project context based on contextual factors (Iivari, 1989). A contingency factor approach is therefore more about selection of a complete method from a portfolio of methods than combining practices from multiple methods. Thus, a contingency factor approach would likely work poorly in this agile context. This further supports previous statements that contingency factor approaches often wrongly assume that there exist some predefined method that can support all the needs of the development context (Fitzgerald, Hartnett, et al., 2006; Kumar and Welke, 1992). Another notable difference from a contingency factor approach was that tailoring was not done by strictly considering a set of contextual factors such as team size, team distribution, or contract type (Kalus and Kuhrmann, 2013). Tailoring did consider aspects such as distance between the teams and the customer, for example by the introduction of daily stand-up meetings to ensure adequate customer involvement. However, this was done more in discretion and based on experience from past projects, and experiences made on challenges and needs throughout the course of the project. This is therefore consistent with previous findings that teams in practice often tailor methods based on past experiences (Fitzgerald, Hartnett, et al., 2006; Patel et al., 2004). The fact that the teams had much previous experience and competence may have been an important factor for the resulting tailored method to work well for the teams. Turk et al. (2002) argue that a high level of agility largely depends on adaptive methods and experienced individuals who know how to effectively tailor these. Based on the above discussion, we have thus learned that a more pragmatic approach to method tailoring was used, where an agile method was tailored based on experiences and what seemed to be most practical and needed in the current project situation.

By further comparing the results towards theory on method engineering we see some more similarities in characteristics. Method engineering is the second overarching approach for conducting method tailoring proposed in literature. An interesting remark is that method engineering approaches are also the most extensively used according to a recent comprehensive literature review (Campanelli and Parreiras, 2015). Part of the approach to tailoring in the project was to combine agile practices into a context-specific method that served the teams' actual needs. This can be compared with a method engineering approach, which also proposes that methods should be more or less made ground-up to better match the actual needs of a specific development situation (Conboy and Fitzgerald, 2010; Kumar and Welke, 1992). However, there are quite a few differences from the results in this study and method engineering approaches proposed in literature. Common for both contingency factor approaches and method engineering is that they are based on deductive and theoretical arguments in research (Fitzgerald, Russo, et al., 2000). Method engineering approaches in the literature assume that methods are assembled by pre-defined and pre-tested method fragments (Kumar and Welke, 1992). A method fragment is here a term used to refer to some distinctive part of a method, such as a practice (i.e. activities to be performed, tools, techniques, principles, etc.) (Aydin et al., 2004). Part of the approach to tailoring in the project was described by some informants as a "toolbox" that they used as needed. We learned that this meant that agile practices were added, modified or removed throughout the course of the project. Thus, this is somewhat similar to method engineering in the sense that individual practices were combined into an agile method. However, a notable difference from method engineering in theory was that practices were not conceptualized as such distinct fragments that were pre-defined and used thereafter. This supports previous findings that the concept of agile method fragments may often be oversimplified in research (Fitzgerald, Hartnett, et al., 2006). This is illustrated by the fact that practices were used and tailored to tasks and situations in the different teams. For example, practices such as pair programming was not followed slavishly, but used on occasion when it seemed useful for developers. In general, the tailored method and its underlying practices was more nuanced and used more organically than merely a combination of pre-defined fragments. This is also another confirmation that tailoring was done in a more pragmatic fashion.

By further discussing the results in light of method engineering, there are a few more identifiable differences. Method engineering is proposed as a process that should be performed by a dedicated method engineer who uses computer-aided tools to assemble an appropriate method for a best possible outcome (Brinkkemper, 1996; Campanelli and Parreiras, 2015; Henderson-Sellers and Ralyte, 2010). As we learned from the results, this was not done in the project. Method engineering is characterized as a structured and formal approach that proposes guidelines and structured processes for tailoring. This has been criticized for often being both resource- and time-consuming in practice, and unwelcome in some contexts as a result (Fitzgerald, Hartnett, et al., 2006; Fitzgerald, Russo, et al., 2000). This is arguably particularly true for agile contexts which emphasize people and interaction over processes and tools (Fowler and Highsmith, 2001). It is therefore not surprising that method tailoring in the studied agile project instead was done in a more informal and unstructured manner. There was no dedicated role for method tailoring, although the project manager played an important role, and nor was there a defined and structured tailoring process. Instead, tailoring seemed to take place as an informal sideline activity. An informal form of tailoring have often been observed in practice in previous research, and especially in smaller organizations or projects where it is not desirable to spend much time on dedicated method tailoring processes (Pedreira et al., 2007). Overall, the approach to method tailoring was thereby characterized as an informal and less structured activity than what some literature suggests. This appears to be common also in more traditional agile project contexts (e.g., Conboy and Fitzgerald, 2010; Fitzgerald, Hartnett, et al., 2006).

Moreover, there was much flexibility in which practices each of the teams used and how they were used. At a team level, ongoing ad hoc tailoring of practices was done without the project manager being directly involved. The fact that teams could make ad hoc adaptations and self-organize the use of practices can potentially have negative effects such as management being unable to ensure conformity in the method and that the level of compliance is different among teams (Conboy and Fitzgerald, 2010). However, an interesting finding is that the flexibility that the teams were given seemed beneficial because they could tailor practices to the tasks at hand, which was usually quite different from team to team. This is also emphasized in the principle of simplicity in the agile manifesto (Fowler and Highsmith, 2001, p. 5): "Include only what everyone needs rather than what anyone needs, to make it easier for teams to add something that addresses their own particular needs". No findings indicated that ad hoc tailoring on a team level led to obvious negative results in method, and there was a general consensus among the informants that the method worked very well. There were some challenges in the method that they worked to improve, but this seemed to have more to do with factors such as organizational culture than issues with the flexibility in the method. It is however possible to assume that the informal and ad hoc tailoring on the team level worked well because the teams consisted of mostly experienced and competent developers and designers who had much experience from agile projects. If this had not been the case, this could have required a more structured and formal approach to method tailoring to give good results (Pedreira et al., 2007). Another factor that very likely also contributed to satisfactory results was that some guidelines were put in place for the teams by the project manager to ensure some similarity in the method and in practices to ensure some overall governance. For example, there were committed use of Kanban boards, self-organizing and cross-functional teams, daily stand-up meetings, and common stand-up meetings among all teams and team members. Some directions were also given on how to use practices, such as how to set up the the Kanban boards.

Although method tailoring was done on an informal and ad hoc basis on team level, the results depict tailoring as a conscious, continuous, and important activity in the project. Tailoring was necessary as a continuous improvement effort to maintain and even increase productivity and delivery speed throughout the course of the project. One example is that DevOps and Continuous Deployment was later introduced to the method to improve both the speed and frequency of deliveries. An important responsibility for the project manager was to make continuous evaluations as to whether the method covered their needs, to make adjustments when challenges or improvement potential was identified, and in general work towards optimizing the method. Except for tailoring on a team level being more ad hoc, method tailoring was therefore largely characterized as a purposeful and disciplined activity from the project manager's point of view. According to Conboy and Fitzgerald (2010), purposeful and disciplined method tailoring is important for effective method tailoring. The project manager discussed methodological aspects with committed team members and also with the customer. In conjunction with the introduction of agile practices, such as self-organizing teams, it was important to develop the right organizational culture for an agile environment together with the customer. Tailoring can therefore also be understood as a collaborative activity in this case. There seemed to be a culture in place where people used an open dialogue and dared to speak out on issues and improvement measures. Historically, such feedback from team members had led to several successful adjustments in the method, such as changes in how stand-ups were practiced. Had such a culture not been in place, it was thought to require more formal arenas like retrospectives in Scrum (Schwaber, 2004) where you specifically discuss process related improvement measures. As have been discussed, it was important to have the project manager carry out continuous and careful assessment and fine-tuning of the method. This was done in collaboration with he customer and the teams to ensure they achieved their goals and were able to deliver good results.

5.1.2 Combination of Agile Practices

The second and last research question was given as follows:

> **RQ2:** How were practices combined into an agile method with aim towards continuous software delivery?

As a further contribution to answering the problem statement, it was interesting to investigate the actual tailored agile method in the project more closely. For this, I have focused on identifying the prominent agile practices used in the project and how these were combined and tailored to enable the teams to deliver changes and new software features on a continuous basis. The following discussion will mainly focus on the results on agile practices from Section 4.2.2. These will be discussed against the theoretical basis provided throughout Chapter 2. Some of the other results may also be used to supplement in some places.

As we learned from the results and previous discussion, agile method tailoring was essentially done by combining agile practices from various sources into an agile method that met the actual needs of the current project situation. This is interesting, because this confirms the importance of method tailoring in practice and illuminates the motivation for method tailoring research. It was apparent that a practice was typically perceived as valuable if it covered their needs and contributed to least possible overhead in the method while still maintaining just enough formalities and control. Thus, it can be understood that an important aspect of tailoring with aim towards continuous software delivery was to omit unnecessary time-consuming practices that would have reduced delivery speed. The result of this seemed to be a very lightweight agile method with very limited regulations. This is very much in line with agile principles, where one ideal for agile methods is that it is minimalist and do not prescribe unnecessary or time-consuming practices that do not add value to the product and elongate the development process (Fowler and Highsmith, 2001; Highsmith and Cockburn, 2001). Working according to agile principles were explicitly stated to be important for the suppliers in their given context. There was a high rate of change and innovation going on, which made agility particularly important. When there are changing needs, agility must be incorporated by deliberately designing lightweight methods that are amenable to tailoring (Henderson-Sellers and Serour, 2005). Additionally, when there are turbulent business environments like in this case, the change tolerance of the method should be geared to the rate of change in the specific project environment for adequate agility (Highsmith and Cockburn, 2001). These aspects seemed to have been important considerations when the agile method was tailored in the project. The method was very minimal and lightweight, with exceptionally few formalities and regulations. The results indicate that all the agile practices seemed to contribute positively to agility and not hinder the teams, which is important in agile methods (Conboy, 2009). That software was delivered continuously show that the method was geared to the high rate of change in the project. Also, since they did not strictly and formally confine to method frameworks and prescribed few rules they were largely adaptive (Kniberg and Skarin, 2010). Thus, it appeared that one important aspect of the combination of practices was to achieve a minimal method with only the necessary practices. Had the method contained many more time-consuming and formal practices it is possible to assume that it could have hindered overall agility and the software delivery rate in the project. Furthermore, we will discuss the actual prominent practices in the minimal agile method and how these were important.

The most fully adopted agile method in the project was Kanban, although they were not formally bounded by it as a framework. Kanban boards were used to manage and visualize work and progress for individual tasks all the way from the backlog to deployment. This included putting limits on work-in-progress (WIP), and monitoring flow with a cumulative flow diagram (CFD). Thus, there were a relatively high degree of adherence to Kanban's three practices (Kniberg and Skarin, 2010). The use of Kanban implied that a flow-based development model was adopted in the project (Birkeland, 2010). Flow-based development implies that as soon as a feature is identified, it can be developed, tested and deployed immediately rather than being batched and completed as a larger and more comprehensive delivery. This also requires establishing a process that not only focuses on development in isolation, but an end-to-end concept that also takes into account other aspects such as planning, deployment, and operation (Fitzgerald and Stol, 2014). Thus, a continuous flow of software features can be delivered and delivery time of individual features can be reduced. A flow-based development model with Kanban was therefore clearly an important element of the method in the project. In contrast, an iterative and sprint-based model like in Scrum would have been less compatible in this context. The choice to adopt Kanban rather than Scrum in project contexts where there is a need to deliver continuously is in line with previous observations in research (Fitzgerald and Stol, 2014). We learned from the results that the teams broke down tasks as small as possible and represented these as individual cards in respective boards. This resulted in a normalization in task size, which ultimately led to a relatively stable flow of tasks being completed. Breaking down tasks into small work packages makes it easier to limit WIP, which in turn results in faster deliveries when using Kanban (Ahmad et al., 2013). The fact that the Kanban boards were digital and contained many configuration options made ongoing customization of boards convenient, such as adding and removing columns to optimize flow and overview. By setting up multiple boards in the different teams for different solutions allowed a separation of non-related tasks, and likely improved overall flow and visibility.

Furthermore, establishing a flow-based development model as an end-to-end process required the teams to gain control over production environments and deployment processes. This included an introduction of DevOps which entailed that the developers took over responsibility for what was previously separate and externally managed deployment and operation. The purpose of DevOps is to align the incentives of both development and operations (Humble and Molesky, 2011), and is an important concept in today's context of pursuing continuous software delivery (Fitzgerald and Stol, 2014). In the studied project, this was achieved by merging the operations role into the developer role, which again introduced additional tasks and responsibilities for the developers in the project. This is consistent with earlier observations that introduction of DevOps tend to force a rethinking and reorientation of roles in development and operations activities (Lwakatare et al., 2016a). A prerequisite for the developers' ability to handle both deployment and operation as part of the daily development activities was to move existing and new systems to a cloud-based Platform as a Service solution (PaaS). Thus, operation of their applications could be done as a sideline activity to development in a less resource- and time-consuming manner by avoiding cumbersome tasks such as managing operating systems and infrastructure. DevOps gave the developers more control of their own applications and improved their ability to handle application problems. This have been observed to be one important advantage with closing the gap between development and operation, which in turn helps to broaden developers' skillset and knowledge (Lwakatare et al., 2016a). This control may in turn enable single teams to alone be responsible for all aspects of development and operations for all systems and services (Lwakatare et al., 2016a) which was largely the case in the studied project. Perhaps most important, DevOps made it possible to introduce Continuous Deployment as a practice by automating integration, builds and deployment processes. Such automation is an important dimension in DevOps to accelerate delivery of change (Lwakatare et al., 2016a). With control over and automation of these processes, a continuous flow of tasks from end-to-end could be achieved, which clearly had good synergy with the adopted Kanban approach.

In general, agile practices can be grouped into the three groups (Lee and Yong, 2013); management practices like in Kanban, software process practices like continuous and incremental delivery, and software development practices like Continuous Deployment. Practices within all these groups were both needed and adopted in the project. As we know from the literature, Kanban only contain management practices for managing and visualizing work flow (Kniberg and Skarin, 2010). As a result, it was necessary for the teams to also introduce other needed management practices such as daily stand-up meetings for customer involvement and coordination.

A significant factor for customer involvement was that the teams were collocated with the customer in the same premises. This enabled very active customer participation through daily involvement of the customer in multiple arenas, including daily stand-up meetings. With daily involvement of the customer, the teams could also ensure that the customer was updated on what was happening in development and could receive very frequent feedback and verification of individual completed tasks. In general, an increased customer focus through frequent feedback from the customer and dynamic prioritization of features are key factors in agile software development (Highsmith and Cockburn, 2001). The high amount of customer involvement in the project shows this, and was considered crucial for utilizing the benefits of continuous software delivery. For example, planning was not done for longer periods and there was very little time spent on estimation of tasks in general. Instead, planning and prioritization of tasks was a daily activity in collaboration with business developers and by actively using the digital Kanban boards as a tool to facilitate this. This may be understood as a form of continuous planning which can ensure continuous alignment between the needs of business and software development in contexts where software is delivered continuously (Fitzgerald and Stol, 2014). Furthermore, we learned from the results that there were some challenges regarding adequate customer involvement in the project. Utilizing continuous software delivery seemed to benefit largely from also having customers available at all times. There was some distance between the teams and the customer representatives, which created a barrier for continuous involvement. Vinekar et al. (2006) argues that agile methods depend on an on-site customer to identify and prioritize features, provide feedback, and guide change throughout development. Extreme Programming (XP) prescribes such an on-site customer practice where the customer works as part of the team to do these tasks for the team continuously (Beck, 2004). This was not done in the project, but it was argued to be a potential improvement to introduce a practice similar to the one in XP where the business developers worked as part of the teams. Ensuring adequate customer involvement is an important success factor in agile projects (Hoda et al., 2011; Misra et al., 2009). To ensure adequate customer involvement, the suppliers therefore had to introduce practices such daily stand-ups and chat to meet their needs for daily involvement, in addition to weekly meetings with other key customer representatives for more overall reporting of progress and planning.

Furthermore, it was important to ensure coordination both across and within the teams. The results showed that a practice called common stand-ups was later introduced as a weekly meeting where all of the teams including management and advisors participated. Common stand-up meetings enabled everyone to synchronize as a whole team to get a more complete overview and to share knowledge and insight across the teams. This meeting to some extent resemble a Scrum of Scrums event that have been found to be effective when there is a need for inter-team coordination in Scrum projects with multiple teams (Dingsøyr, Moe, et al., 2017). Daily stand-up meetings were tailored to be somewhat different than in the textbook version of Scrum. For example, daily stand-up meetings were held in an open format and was very centered around the Kanban boards. The customer was also involved in the daily stand-up meetings, which is not part of the prescribed practice in Scrum. The fact that daily stand-ups had an open format meant that discussions about various important issues could occur, but with one side effect being that conversations were sometimes less focused. Stand-ups were held standing and were supposed to be timeboxed for 15 minutes as prescribed in Scrum (Schwaber, 2004), but could sometimes last longer than intended especially when the customer was involved. Some unfortunate effects could potentially have be mitigated by management giving stricter guidelines for how the daily stand-up should be practiced. At the same time, it can be desirable to give the teams some flexibility to use and tailor practices to best suit their tasks, situation and preferences (Fowler and Highsmith, 2001). Furthermore, team's being collocated in an open work area enabled continuous oral dialogue and ad hoc coordination and collaboration with other teams when needed. All in all, active use of coordination practices were largely needed to ensure continuous management of dependencies and ensure transparency within and across teams.

One last very prominent agile practice was that the teams were largely self-organizing (i.e. self-managing) and cross-functional. Self-managing teams offer potential advantages over traditionally managed teams because they bring decision-making authority to the level of operational problems and uncertainties and thus increase the speed and accuracy of problem solving (Moe et al., 2009). Self-organizing teams that encourage role interchangeability are considered to be important in agile software development (Nerur, Mahapatra, et al., 2005). Moreover, cross-functional teams possesses the skills necessary to carry out its tasks without external support, and is appreciated in agile methods such as Scrum (Schwaber, 2004) and XP (Beck, 2004). We learned from the results that the teams had become more cross-functional and self-organizing over time. In order to increase the degree of autonomy, it was necessary to support more decision-making authority at the team level. This involved much organizational- and cultural development in close collaboration with the customer to move decision-making authority closer to the team-level. According to Moe et al. (2009), the performance of self-managing teams not only depends on the competence of team members to manage and execute their work, but also on the organizational context that management provides. Organizational culture is a known challenge related to agile project contexts that may affect decision-making processes and problem solving strategies (Gandomani et al., 2013; Nerur, Mahapatra, et al., 2005). This is apparent also in the results, and illustrates why agile method tailoring can be particularly challenging as it must consider also aspects related to customers and stakeholders and complexities within their organization (Conboy and Fitzgerald, 2010). Moreover, although there were multiple operative self-organizing and cross-functional teams working with different tasks, we learned that an important principle was to still have shared responsibility and goals as a whole team. This likely had a more indirect impact on performance, but contributed to overall transparency and resource utilization. One main goal of this principle, and related practices such as common stand-up meetings, was to ensure that people had overall insight and that people could work across areas. This is an important factor in self-managing teams to ensure that available personnel can be flexible and utilized across areas when there are changing environments (Moe et al., 2009). An interesting side note is that the principle of one whole team in the project also share similarities with the Lean principle of "seeing the whole", which was briefly mentioned in the literature on Kanban (Section 2.2.3). Focusing on seeing the whole when there are many teams can prevent that only parts of many software solutions are optimized on the expense of the whole, thus increasing overall quality (Poppendieck and Poppendieck, 2003). This was arguably important in the project, where one important purpose was to increase overall quality of solutions through more cohesion and unified management among these. Overall, facilitating cross-functional and self-organizing teams enabled many important decisions to be made and completed quickly by the teams themselves, such as verification and deployment of tasks, which again contributed to faster deliveries.

The purpose of the overall agile method was to make short time-to-market possible and to help meet the customer's goals of offering better products and services and stay competitive in a challenging business environment. Throughout this study, it was apparent that there were many factors and prerequisites for enabling this through continuous software delivery. This study has focused on the methodological aspects of this. However, other factors were also clearly important in this context. Some have been briefly been touched upon, such as focusing on a close connection between business and development (Fitzgerald and Stol, 2014) and technological factors such as using mature technology (Dingsøyr and Lassenius, 2016). A concluding remark is that it was the combined effects of many factors, including the tailoring of the agile method discussed, which made it possible for the teams to deliver continuously, promptly, and with the right priorities.

5.2 Implications of the Research

This section presents what I regard to be important contributions of this study and their implications for practice and the research field.

One main contribution of this research is a thorough description of an agile project from practice. The amount of empirical research on the topics that have been covered in this thesis is currently very limited, which the literature review and requests by researchers shows. This also means that only a limited amount of empirical research is contributing back to practice. I therefore believe this study will be valuable to other researchers and students, and not least to practitioners who seek information about method tailoring and agile methods and practices. I would argue that the agile method in the case represents an example of a state-of-the-art method, which also seemingly worked very well and provided good results in the project. This should arguably make this a particularly interesting case.

Implications for practice

The thorough case description and the findings included in this thesis should provide a basis for comparison with other project contexts. Practitioners may do this to evaluate whether a similar approach to method tailoring and agile development may be applicable in other situations. However, one should be careful to make assumptions when doing so. There are many situational factors that come into play when tailoring agile methods. For example the level of experience among team members, which was very high in this case. The focus of this study was not to thoroughly evaluate what the prerequisites were for effective tailoring, and thus what exactly made a successful tailoring approach and resulting agile method is somewhat uncertain.

From what I have learned throughout this work, I believe there is reason to suspect a general lack of awareness around method tailoring in software development projects. Some form of method tailoring most likely always happens, since project methods are not universally applicable in their original, textbook format due to varying situational factors. However, the literature indicates that this is often not done in a purposeful and disciplined manner which may cause unfortunate or suboptimal results. The findings in this study shows that method tailoring was an ongoing and important activity which likely had very positive impact on the results in the project. Practitioners may increase their awareness of the importance of method tailoring by reading this thesis, and then further seek out other information to learn how this may be done.

A particularly interesting finding in the case was that the project method was not strictly defined based on an existing method, but was customized for the project by combining agile practices from many sources. This seemed to provide much flexibility and adaptability and thus contribute to overall agility. In many other projects, a method such as Scrum is selected and strictly committed to throughout the project duration. This may even be defined in contractual agreements with customers and stakeholders prior to project start. Some tailoring may still be done, but you are automatically more limited. It is possible to speculate that this may potentially also be a root cause for further lack of awareness of method tailoring because the method is already set in stone or clearly documented. In the studied project there seemed to be little enforcement regarding which project method to use, and the suppliers seemed to be very aware that this ultimately enabled more flexibility to adopt practices and adjust to whatever worked best in the current situation. More practitioners may find a similar method tailoring approach useful. However, this will require knowledge of more than one agile method and how these may be combined to complement each other. This may therefore require that organizations invest in adequate training in agile methods and practices for managers and developers. This can be costly, but may lead to more optimized agile methods and consequently better project results. It is worth noting that it may also require other prerequisites, such as incorporating more flexibility in method definitions in contracts.

Furthermore, it can be argued that studying method tailoring in the given agile software development context does not necessarily mean that the findings can not be applied to more traditional contexts. This may for example be agile projects that does not necessarily pursue continuous software delivery. Fundamental concepts in the findings in this study may even be generalizable to non-agile projects or projects that involves other types of product development. For example the idea of picking and choosing practices from multiple project methods. However, what exactly is generalizable is hard to say as it would highly depend on the context, and as a result I have tried not to make many generalizations in this single-case study. More on this in the last section of this chapter on research limitations (Section 5.3).

Implications for research

This study contributes with a new addition of research on method tailoring in the context of agile software development projects. This study stands out from much previous research in the sense that method tailoring was studied in a modern development context where there was a need for continuous software delivery. Past research have investigated method tailoring in arguably more traditional agile contexts (e.g., Conboy and Fitzgerald, 2010; Fitzgerald, Hartnett, et al., 2006) and in large-scale agile projects (Rolland et al., 2016). This research provides a view on method tailoring in another and arguably very interesting context.

Past method tailoring research has focused much on tailoring approaches. This is natural since tailoring approaches define the actual procedure and guidelines for how effective tailoring can be conducted. However, more research on this is still required, which is also why it was recently called out as a possible research direction (Campanelli and Parreiras, 2015). Throughout this study, we have learned that the current suggestions for approaches to method tailoring in research is mostly designed from deductive and theoretical propositions. They are often very structured and formal, and defines processes and guidelines that have been observed to be time-consuming in practice. As a result, these have been criticized for limited applicability in practice and for containing several challenges (Campanelli and Parreiras, 2015; Conboy and Fitzgerald, 2010; Fitzgerald, Russo, et al., 2000). The approach to method tailoring identified in this study had some similarities to existing approaches, but was also characterized to be much more informal, lightweight and less process oriented than the two established approaches. In relation to this, the discussion also argued that a very heavy and time-consuming process would not be very compatible in an agile context like this. Thus, there is ground to further support the criticism of existing approaches, and why these are not well suited in many contexts such as modern agile projects. However, this is not very surprising because the two established overarching approaches to tailoring predates agile software development and was thus not designed with agile values in mind. I believe this should be acknowledged in future research on method tailoring in agile contexts, and that more attention is directed at designing alternative tailoring approaches better suited for agile contexts. More on this in suggestions for future research (Section 6.2).

5.3 Research Limitations

The method chapter presented tactics that were used to improve the overall validity and quality of this research (Section 3.10). This section will address factors that may still have had an impact on the result, and which constitute the limitations of the research.

Research methodology:
This research was conducted as a single-case study. This imposes limitations regarding the generalizability of findings (Lee and Baskerville, 2003). Some generalization to theory have been done to improve the overall validity (Yin, 2014), but the findings and inferences that have been made are still limited to a single project context and not a broader collection of cases. However, the single-case study provides in-depth insight and represents an exemplar that may still hold much valuable information for many.

Data collection methods and collected data:
I had no previous experience with carrying out case study research of this scope. As a result, the implementation of the chosen data collection methods was a possible limitation. I had little to no previous experience with interviews or observations with the aim of collecting qualitative data. Case study research requires experience, and little experience may negatively impact the data collection method implementation and thus the collected data (Yin, 2014).

Data collection was mainly conducted in two occasions using interviews, observations and some analysis of documentation. The collected data was therefore mainly qualitative data. By integrating additional techniques for data collection, such as a survey for quantitative data collection, my understanding of the case and findings could have been improved (Runeson and Höst, 2008). However, due to constraints on resources this was not prioritized. Again, it should be clarified that a survey was used during the interviews, but that this was not intended for comprehensive quantitative data collection, but rather to guide some of the qualitative data collection. This was described in the presentation of the results (Section 4.2.2).

Another limitation was that I did not manage to ask every informant all the same questions, although the interviews were quite long lasting. This was due to several factors. The interviews were semi-structured and time was spent asking questions of interest outside of the interview guides which sometimes made scheduling the questions challenging. Also, the informants had different roles and thus different prerequisites for answering some of the questions, which could better have been accounted for when preparing the interview guides. Overall, this limited the amount of data on some concepts to draw inferences from, and may thus have impacted the quality of some inferences.

In addition to constraints on time, there were limited available informants who could participate in interviews. This could potentially affect data quality negatively, since only some viewpoints was covered in the data. Having more data from more informants would have further increased the validity of inferences from the data. Optimally speaking, interviewing more or all members of the project, including the customer, would improve research quality. This was, however, not possible due to constraints on resources.

Scope of theory and discussions:
In addition to little experience with conducting research of this scope, I also only had limited overview of the area of research. The scope and depth of the discussion, and the completeness of the included literature and theory, may therefore have been compromised on some levels compared to the work from experienced scholars or researchers (Runeson and Höst, 2008). However, mine as well as other researchers impression is that there is a very limited amount of research on some of the topics studied in this thesis. Most critical is perhaps the seemingly lack of empirical research on method tailoring in agile software development contexts. As a result, this study is automatically limited to the relatively scarce theoretical basis that exists on these core topics.

Furthermore, when this work was initially started a suitable case to be studied was not yet identified. This came into place only a short time later. Thus, all work related to data collection and transcription of the data was done within the limited timeframe of the study. Had I been able to start this work earlier, more effort could have been spent on literature reviews, analysis, discussion, and writing in general.

Conclusion | 6

This chapter presents the conclusions of the study with regard to the problem statement and the underlying research questions. Proposals for future research is provided in the end.

6.1 Problem Statement

This study has aimed at empirically investigating how agile methods can be tailored for software development projects where there is a need to deliver software on a continuous basis. As we learned in the introduction and the theory chapter (Section 2.1.3), agile methods have enabled companies to embrace change by delivering working software in iterations typically every two to four weeks. However, in today's challenging business context, practitioner's have increasingly started to adopt and tailor agile approaches with the aim of software delivery up to multiple times a day to reap the benefits of even greater agility. In the problem description (Section 1.2) this was viewed as a trend towards *continuous software delivery*. This master thesis has showcased a project which through tailoring of agile methods and practices had taken agile software development this one next step towards continuous software delivery. Thus, this represented a great opportunity to investigate the following problem statement:

> *How can agile methods be tailored with aim towards continuous software delivery in projects?*

Exploring this towards the case have led to much new and valuable insight into agile method tailoring in a modern software development context. For the purpose of this research, the problem statement was narrowed down into two more specific research questions. The following sections will conclude the findings of the study for each research question respectively. This is done by looking back on what we have learned, with emphasis on the discussion in the previous chapter.

RQ1: What characterizes an agile method tailoring approach in a modern software development context?

Several prominent characteristics on how agile method tailoring was conducted in the project was identified. These can be summarized as follows:

- Agile method tailoring was done without confining to a method framework, but by being flexible in combining agile practices from multiple methods or stand-alone practices that contributed to agility.
- Selection and combination of practices took place in a pragmatic fashion, by selecting agile practices that covered their situational needs based on previous experience and experiences made throughout the course of the project.
- Tailoring was conducted in an informal and less structured manner, and did not involve rigid and time-consuming method tailoring processes and guidelines as used in some organizations. Thus, the tailoring approach complied with agile values to a much greater extent.
- Ongoing ad hoc tailoring of practices was done by each individual team. The teams were largely self-organizing and had the authority to select and tailor their own practices to tasks, situations and preferences. It was, however, important that management prescribed some common practices and guidelines in the method and practices across teams to ensure adequate overall governance.
- Agile method tailoring was an important, purposeful and continuous activity, in which management made improvements in the method based on own assessments and feedback from committed team members. Close collaboration with the customer to facilitate for an appropriate agile environment was important in conjunction to this.

Many of the above characteristics could be recognized also in existing research on method tailoring in more traditional agile contexts. Perhaps the most prominent characteristic that differed from previous findings in other case studies was the first item; that tailoring was not confined to a specific set of method frameworks. This seemed to contribute to a high level of adaptability, which is arguably important in business environments with high rates of change.

RQ2: How were practices combined into an agile method with aim towards continuous software delivery?

The project context involved a high degree of change and innovation, which was handled by achieving a high level of agility and a short time-to-market. In this regard, it was important to tailor a minimal agile method by omitting practices and formalities that were not deemed necessary or which hindered the teams from delivering software on a continuous basis. The prominent agile practices were the following:

- Extensive use of Kanban boards for visualization and management of tasks. This implied a flow-based development model as an end-to-end process in which individual tasks were prioritized, developed, tested, verified and deployed independently and continuously. This contrasts an iterative and time-based model (e.g. sprint-based) in which tasks are completed in batches over a period of time before it is delivered.
- Establishing an end-to-end process required introducing DevOps in which development and operation was an integrated process. This enabled the introduction of Continuous Deployment, which involved automation of integration and deployment processes all the way to production for easy and quick delivery of changes and new features.
- Collocation of the teams and the customer enabled daily customer involvement. This made it possible to practice continuous planning and receive frequent feedback and verification of individual tasks from customer representatives within several arenas.
- Multiple arenas for collaboration and coordination within and across teams enabled continuous management of dependencies and transparency. Among these, an open work area and stand-up meetings were regarded particularly useful practices.
- Productive self-organizing and cross-functional teams who had the authority and capability to make quick decisions on tasks, and possessed the necessary competence to execute their work. This in turn contributed to rapid delivery of change and new software features.

In addition, other agile practices were used as needed within the teams. These were mentioned in the results (Chapter 4). Each practice had a role in the method, whether it was for management, process, or more technical development purposes. The combination made it possible for the teams to deliver continuously, promptly, and with the right priorities.

To conclude, the findings of this research are consistent with existing research that it is important to tailor agile methods to the actual needs of the project context. The findings suggest that method tailoring should be practiced as a continuous improvement effort to ensure adequate agility is maintained and improved and that changing needs are met throughout the course of a project.

6.2 Future Work

An important part of exploratory case studies is the discovery of new directions for further research (Runeson and Höst, 2008). Following are the suggestions for future work based on what have been learned from this study.

Findings from this study showed that agile method tailoring can take place as an informal, ad hoc and unstructured activity and still produce good results. Similar have also been observed in other agile projects (e.g., Fitzgerald, Hartnett, et al., 2006). In other cases, however, it has been suggested that such an approach to agile method tailoring can also have negative impact on the results of tailoring efforts and other side effects (e.g., Conboy and Fitzgerald, 2010). In other words, a lack of guidelines and structure on how to effectively tailor agile methods and practices to a situation can be a problem, in particular when there is lack of knowledge and experience (Conboy and Fitzgerald, 2010). In the discussion it was argued that one possible success factor for tailoring in the studied case was the fact that the teams consisted of highly experienced and competent people, many of which had worked with similar agile practices in the past. However, this is likely not the case in many project contexts. In general, there are few proposed approaches to guide effective tailoring to date, and especially not for agile methods. The most commonly used approach, method engineering (Campanelli and Parreiras, 2015), is known for being both resource- and time-consuming in practice, and was not initially designed with agile values in mind. Based on these matters, it is therefore interesting for future research to further investigate possible method tailoring approaches which take into account agile values and which provide guidelines for effective tailoring. One starting point may be to investigate the use of retrospectives for this, which had been used in the case, and which is proposed as an activity for process improvement in widely used agile methods such as Scrum (Schwaber, 2004).

The second part of this work focused on the combination of agile practices with aim towards continuous software delivery. In conjunction to this, DevOps was highlighted an important concept which is not yet well established in research (Lwakatare et al., 2016a). An interesting finding in this study, although not surprising, was that a flow-based method with Kanban had good synergy with DevOps. Some research has started to appear that have investigated DevOps and its relationship to other agile practices (e.g., Lwakatare et al., 2016b). Future research can further investigate this, and more specifically how these may be tailored and used together in future agile software development projects. This study represents one such contribution of empirical research that show how DevOps can be an important part of agile methods.

Bibliography

Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. (2002). “Agile software development methods: Review and analysis”. In: *VTT Publications*, no. 478, pp. 3–107.

Abrahamsson, P., Warsta, J., Siponen, M. T., and Ronkainen, J. (2003). “New directions on agile methods: a comparative analysis”. In: *Proceedings - International Conference on Software Engineering 2003*. Ieee, pp. 244–254.

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

Atlassian (2018). *DevOps*. [Online; accessed February 8, 2018]. URL: `https://www.atlassian.com/devops`.

Aydin, M. N., Harmsen, F., Van Slooten, K., and Stegwee, R. A. (2004). “An agile information systems development method in use”. In: *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 12, no. 2, pp. 127–138.

Banica, L., Radulescu, M., Rosca, D., and Hagiu, A. (2017). “Is DevOps another Project Management Methodology?” In: *Informatica Economica*, vol. 21, no. 3, pp. 39–51.

Baskerville, R., Pries-Heje, J., and Madsen, S. (2011). “Post-agility: What follows a decade of agility?” In: *Information and Software Technology*, vol. 53, no. 5, pp. 543–555.

Beck, K. (1999). “Embracing change with extreme programming”. In: *Computer*, vol. 32, no. 10, pp. 70–77.

Beck, K. (2004). *Extreme programming explained: embrace change*. 2nd ed. The XP series. Boston: Addison-Wesley.

Birkeland, J. O. (2010). “From a Timebox Tangle to a More Flexible Flow”. In: *XP 2010*. Springer, pp. 325–334.

Brinkkemper, S. (1996). “Method engineering: Engineering of information systems development methods and tools”. In: *Information and Software Technology*, vol. 38, no. 4, pp. 275–280.

Brooks, F. (1987). “No Silver Bullet: Essence and Accidents of Software Engineering”. In: *Computer*, vol. 20, no. 4, pp. 10–19.

Brown, A., Forsgren, N., Humble, J., Kersten, N., and Kim, G. (2017). *2017 State of DevOps Report*. Report. URL: `https://puppet.com/resources/whitepaper/state-of-devops-report`.

Campanelli, A. S. and Parreiras, F. S. (2015). "Agile methods tailoring – A systematic literature review". In: *The Journal of Systems and Software*, vol. 110, pp. 85–100.

Cockburn, A. (2002). *Agile software development*. Addison-Wesley Boston.

Cohen, D., Lindvall, M., and Costa, P. (2004). "An Introduction to Agile Methods". In: *Advances in Computers*, vol. 62, pp. 1–66.

Conboy, K. (2009). "Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development". In: *Information Systems Research*, vol. 20, no. 3, pp. 329–354.

Conboy, K. and Fitzgerald, B. (2010). "Method and developer characteristics for effective agile method tailoring: A study of XP expert opinion". In: *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 1, pp. 1–30.

Corona, E. and Pani, F. E. (2012). "An investigation of approaches to set up a Kanban board, and of tools to manage it". In: *Proceedings of the 11th International Conference on Telecommunications and Informatics (TELEINFO'12) and the 11th International Conference on Signal Processing (SIP'12), Saint Malo, France*, pp. 7–9.

Dingsøyr, T. and Lassenius, C. (2016). "Emerging themes in agile software development: Introduction to the special section on continuous value delivery". In: *Information and Software Technology*, vol. 77, pp. 56–60.

Dingsøyr, T., Moe, N. B., Fægri, T. E., and Seim, E. A. (2017). "Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation". In: *Empirical Software Engineering*, pp. 1–31.

Dingsøyr, T., Nerur, S., Balijepally, VG., and Moe, N. (2012). "A decade of agile methodologies: Towards explaining agile software development". In: *The Journal of Systems and Software*, vol. 85, no. 6, pp. 1213–1221.

Dybå, T. and Dingsøyr, T. (2008). "Empirical studies of agile software development: A systematic review". In: *Information and Software Technology*, vol. 50, no. 9-10, pp. 833–859.

Dybå, T. and Dingsøyr, T. (2009). "What do we know about agile software development?" In: *IEEE software*, vol. 26, no. 5, pp. 6–9.

Fitzgerald, B., Hartnett, G., and Conboy, K. (2006). "Customising agile methods to software practices at Intel Shannon". In: *European Journal of Information Systems*, vol. 15, no. 2, pp. 200–213.

Fitzgerald, B., Russo, N., and O'Kane, T. (2000). "An empirical study of system development method tailoring in practice". In: *ECIS 2000 Proceedings*, pp. 4–11.

Fitzgerald, B. and Stol, KJ. (2014). “Continuous software engineering and beyond: trends and challenges”. In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*. ACM, pp. 1–9.

Flyvbjerg, B. (2006). “Five misunderstandings about case-study research”. In: *Qualitative inquiry*, vol. 12, no. 2, pp. 219–245.

Fowler, M. and Highsmith, J. (2001). “The agile manifesto”. In: *Software Development,* vol. 9, no. 8, pp. 28–35.

Gandomani, T. J., Zulzalil, H., Ghani, A. A. A., Sultan, A. B. M., and Nafchi, M. Z. (2013). “Obstacles in moving to agile software development methods; at a glance”. In: *Journal of Computer Science,* vol. 9, no. 5, pp. 620–625.

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

Harmsen, A. F., Brinkkemper, J. N., and Oei, JL. H. (1994). *Situational method engineering for information system project approaches*. University of Twente, Department of Computer Science.

Henderson-Sellers, B. and Ralyte, J. (2010). “Situational Method Engineering: State-of-the-Art Review”. In: *Journal Of Universal Computer Science*, vol. 16, no. 3, pp. 424–478.

Henderson-Sellers, B. and Serour, M. (2005). “Creating a Dual-Agility Method: The Value of Method Engineering”. In: *Journal of Database Management*, vol. 16, no. 4, pp. 1–23.

Highsmith, J. and Cockburn, A. (2001). “Agile software development: The business of innovation”. In: *Computer*, vol. 34, no. 9, pp. 120–127.

Hoda, R., Noble, J., and Marshall, S. (2011). “The impact of inadequate customer collaboration on self-organizing Agile teams”. In: *Information and Software Technology*, vol. 53, no. 5, pp. 521–534.

Humble, J. and Molesky, J. (2011). “Why enterprises must adopt devops to enable continuous delivery”. In: *Cutter IT Journal*, vol. 24, no. 8, pp. 3–39.

Iivari, J (1989). “A methodology for IS development as organizational change: A pragmatic contingency approach”. In: *Systems Development for Human Progress, North-Holland, Amsterdam*, pp. 197–217.

Ikonen, M., Kettunen, P., Oza, N., and Abrahamsson, P. (2010). “Exploring the Sources of Waste in Kanban Software Development Projects”. In: *36th Euromicro Conference on Software Engineering and Advanced Applications*. EUROMICRO Conference Proceedings, pp. 376–381.

Jørgensen, M. (2016). “A survey on the characteristics of projects with success in delivering client benefits”. In: *Information and Software Technology*, vol. 78, pp. 83–94.

Kalus, Georg and Kuhrmann, Marco (2013). “Criteria for software process tailoring: a systematic review”. In: *Proceedings of the 2013 International Conference on Software and System Process*. ACM, pp. 171–180.

Kniberg, H. and Skarin, M. (2010). *Kanban and Scrum: making the most of both*. USA: C4Media.

Kumar, K. and Welke, R. J. (1992). “Methodology Engineering R: a proposal for situation-specific methodology construction”. In: *Challenges and strategies for research in systems development*. John Wiley and Sons, Inc., pp. 257–269.

Lee, A. S. and Baskerville, R. L. (2003). “Generalizing Generalizability in Information Systems Research”. In: *Information Systems Research*, vol. 14, no. 3, pp. 221–243.

Lee, G. and Xia, W. (2010). “Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility”. In: *Mis Quarterly*, vol. 34, no. 1, pp. 87–114.

Lee, S. and Yong, HS. (2013). “Agile software development framework in a small project environment”. In: *Journal of Information Processing Systems*, vol. 9, no. 1, pp. 69–88.

Lindstrom, L. and Jeffries, R. (2004). “Extreme programming and agile software development methodologies”. In: *Information Systems Management*, vol. 21, no. 3, pp. 41–52.

Lwakatare, L. E., Kuvaja, P., and Oivo, M. (2016a). “An Exploratory Study of DevOps Extending the Dimensions of DevOps with Practices”. In: *11th International Conference on Software Engineering Advances*, pp. 91–99.

Lwakatare, L. E., Kuvaja, P., and Oivo, M. (2016b). “Relationship of DevOps to Agile, Lean and Continuous Deployment: A Multivocal Literature Review Study”. In: *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016*. Springer, pp. 399–415.

Misra, S. C., Kumar, V., and Kumar, U. (2009). “Identifying some important success factors in adopting agile software development practices”. In: *The Journal of Systems and Software*, vol. 82, no. 11, pp. 1869–1890.

Moe, N. B., Dingsøyr, T., and Dybå, T. (2009). “Overcoming barriers to self-management in software teams”. In: *IEEE software*, vol. 26, no. 6, pp. 20–26.

Nerur, S. and Balijepally, VG. (2007). “Theoretical reflections on agile development methodologies”. In: *Communications of the ACM*, vol. 50, no. 3, pp. 79–83.

Nerur, S., Mahapatra, RK., and Mangalaraj, G. (2005). “Challenges of migrating to agile methodologies”. In: *Communications of the ACM*, vol. 48, no. 5, pp. 72–78.

Oates, B. J. (2006). *Researching information systems and computing*. London: Sage Publications.

Patel, C., De Cesare, S., Iacovelli, N., and Merico, A. (2004). “A framework for method tailoring: a case study”. In: *2nd OOPSLA Workshop on Method Engineering for Object-Oriented and Component-Based Development*, pp. 1–14.

Pedreira, O., Piattini, M., Luaces, M. R., and Brisaboa, N. R. (2007). “A Systematic Review of Software Process Tailoring”. In: *SIGSOFT Software Engineering Notes*, vol. 32, no. 3, pp. 1–6.

Petersen, K., Wohlin, C., and Baca, D. (2009). “The Waterfall Model in Large-Scale Development”. In: *PROFES*. Springer, pp. 386–400.

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

Rising, L. and Janoff, N. S. (2000). “Scrum software development process for small teams”. In: *IEEE Software*, vol. 17, no. 4, pp. 26–32.

Rodríguez, P., Lwakatare, L. E., Haghighatkhah, A., et al. (2017). “Continuous deployment of software intensive products and services: A systematic mapping study”. In: *The Journal of Systems and Software*, vol. 123, pp. 263–291.

Rodríguez, P., Markkula, J., Oivo, M., and Turula, K. (2012). “Survey on Agile and Lean Usage in Finnish Software Industry”. In: *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, pp. 139–148.

Rolland, K. H., Mikkelsen, V., and Næss, A. (2016). “Tailoring Agile in the Large: Experience and Reflections from a Large-Scale Agile Software Development Project”. In: *International Conference on Agile Software Development*. Springer, pp. 244–251.

Royce, Winston W (1987). “Managing the development of large software systems: concepts and techniques”. In: *Proceedings of the 9th international conference on Software Engineering*. IEEE Computer Society Press, pp. 328–338.

Runeson, P. and Höst, M. (2008). “Guidelines for conducting and reporting case study research in software engineering”. In: *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164.

Schwaber, K. (2004). *Agile project management with Scrum*. Microsoft professional. Redmond, Wash: Microsoft Press.

Scrum.org (2017). *The Scrum Process*. [Online; accessed November 16, 2017]. URL: `https://s3.amazonaws.com/scrumorg-website-prod/drupal/2016-06/ScrumFramework_17x11.pdf`.

Stavru, S. (2014). “A critical examination of recent industrial surveys on agile method usage”. In: *Journal of Systems and Software*, vol. 94, pp. 87–97.

Sutherland, J. and Schwaber, K. (2016). “The Scrum Guide”. In: [Online; accessed November 16, 2017]. URL: `https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf`.

Turk, D., France, R., and Rumpe, B. (2002). “Limitations of Agile Software Processes”. In: *XP2002*, pp. 43–46.

VersionOne (2017). *The 11th Annual State of Agile Report*. Report. URL: `https://explore.versionone.com/state-of-agile/versionone-11th-annual-state-of-agile-report-2`.

Vinekar, V., Slinkman, C. W., and Nerur, S. (2006). “Can agile and traditional systems development approaches coexist? An ambidextrous view”. In: *Information systems management*, vol. 23, no. 3, pp. 31–42.

Webster, J. and Watson, R. T. (2002). “Analyzing the past to prepare for the future: Writing a literature review”. In: *MIS Quarterly*, vol. 26, no. 2, pp. XIII–XXIII.

Williams, L. and Cockburn, A. (2003). “Agile software development: It’s about feedback and change”. In: *Computer*, vol. 36, no. 6, pp. 39–43.

Yin, R. K. (2014). *Case study research: design and methods*. 5th ed. Los Angeles, Calif: SAGE.

Appendices

A

This chapter contain the following attachments:

1. The interview guides that were used to conduct interviews
2. The survey that was used as a starting point for investigating agile practices in the case
3. The template for the confidentiality agreement between the student and the case company

A.1 Appendix A: Interview Guides

Interview guide 1 and 2 for follow on the next page.

Intervjuguide - Runde 1

<u>Forklaringer og fargekoder i guiden:</u>

Blå = hjelpetekst til meg selv (baktanke / oppfølgningsspørsmål / sjekkliste)

Svart = direkte spørsmål

Kursiv = merknad - stilles til utvalgte personer

"Les:" = alternativ betydning / fremstilling av spørsmålet

Intro:

1. Setter i gang for å bruke minst mulig av tiden deres
2. Introduserer meg selv
3. Forklarer hvordan vi gjør intervjuene *(+ et par spørsmål til prosjektleder)*
4. Kjører på med spørsmål
5. Få underskrift på samarbeidsavtale osv.

Presentere meg selv (ca 2 minutt):

Mål: Gjøre meg kjent for informant, og hvorfor jeg er der.

1. Rolle og bakgrunn:
   a. Sivilingeniørstudent NTNU - Studie med vekt på datateknologi og prosjektledelse
2. Kort info om masteroppgaven
3. Hvorfor dette prosjektet var interessant i denne sammenheng
4. Dataen benyttes som utgangspunkt for et casestudie i oppgaven, og mulig jeg setter caset opp mot andre case

Praktisk om intervju (ca 3 minutter):

Mål: Gjøre det klart for informant hvordan intervjuet skal foregå, og hva det innebærer å delta på intervjuet.

5. Tar lydopptak, transkriberes, og alle navn vil anonymiseres
6. *Til prosjektleder:*
   a. Spørre om spesielle ønsker rundt anonymitet
   b. Spørre om videre intervjuer og informasjon (to runder, flere informanter, dokumentasjon)
7. Spørsmål og svar på åpent format:
   a. Svar hvordan du vil, og ingen svar er feil. Er på jakt etter dine personlige meninger og erfaringer.

Introduksjon (5 minutt):

Mål: Bli kjent med informanten.

8. Hva er din rolle i prosjektet?
9. Hva er dine oppgaver og ditt ansvar?
10. Hvor lenge har du jobbet i prosjektet?
11. Intern eller ekstern?

Overordnet forståelse for prosjektet (10 minutter):

Mål: Få god forståelse for og oversikt over prosjektet - produktet/tjenestene, organisasjon, interessenter, mål, overordnet strategi for å oppnå disse målene, osv.

Starter generelt:

11. Beskriv prosjektet for meg (så jeg har noe å starte med).
    a. Mål? (formål, resultatmål; les: hensikt/hva å oppnå)
    b. Produkter og/eller tjenester som leveres?
    c. Hvem er kunden(e)?
    d. Si litt om rammene rundt prosjektet (omfang, tids- og ressursrammer, osv)
    e. Kontrakttype?
    f. Prosjektsteg / faser
    g. Kundekrav og leveransemodell? (skissere denne?)
12. Når begynte prosjektet, og når tror dere målet er nådd?

Organisasjon:

14. Kan du beskrive hvordan prosjektet er organisert? (roller og hierarki, hvor mange?)
    a. *Prosjektleder:* skissere eller sende organisasjonskart?

Oppfølgende spørsmål om ikke dekket i beskrivelser over:

---

15. Er dere ett eller flere team?
    a. Hvis ett team: Teamet er relativt stort (14+ personer) - er det en grunn til at dere organiserer dere som ett team, og ikke deler opp i flere mindre team?
    b. Hvordan fungerer dette?
    c. Kan du beskrive hvordan du og teamet jobber sammen for å oppnå de målene som er satt (illustrer gjerne på papiret).
16. Ledelse:
    a. Kan du beskrive ledelsen og lederrollene i prosjektet?
    b. Hvem tar beslutninger i prosjektet, og hvilke beslutninger tas?
    c. Hvordan formidles disse beslutningene til resten av teamet?
17. Kunde:
    a. Hvordan er kunden involvert i prosjektet?
    b. Hvem er andre viktige interessenter? (sluttbruker, osv.)
    c. Kan du si litt om hvorfor dere gjør det slik? (er det viktig i forhold til metodikken dere bruker?)
18. Arbeidsplass:
    a. Sitter alle, inkludert kunde her?
    b. Er det åpent landskap eller hvordan sitter dere i forhold til hverandre?
    c. Er dette viktig for måten dere jobber på?

Metodetilpasning (20-25min):

Mål: få en god oversikt over metoder og praksiser som benyttes gjennom alle faser i prosjektet. Hvilke tilpasninger er gjort, er dette en bevisst prosess, og hvorfor de er gjort og om de fungerer bra eller dårlig.

20. Har du kompetanse eller erfaring med prosjektmetodikker fra før, eventuelt hvilke?

I de neste spørsmålene vil jeg spørre om beskrivelser rundt metodikk og praksiser i prosjektet. Det er fortsatt fritt frem hvordan du svarer, men sett gjerne svarene i kontekst av hvordan dere jobber metodisk i prosjektet.

21. Med egne ord: hvordan vil du beskrive programvareutviklingsmetodikken dere benytter i prosjektet? (*hvis informant er usikker*: beskriv arbeidsprosessene)

For de metodikkene/praksisene som blir nevnt: *

22. Hvilke “metodikker og underliggende praksiser” benytter dere, smidige og ikke-smidige?
23. Hvordan bruker dere denne metodikken/praksisene (les: arbeidsprosess)?
24. Hvor kommer du inn i denne prosessen?
25. Hvorfor har dere valgt å bruke denne metodikken/praksisene?
    a. erfaring / preferanse?
    b. prosjektkontekst / prosjekttype?
    c. synergier?
    d. hvem tar den beslutningen? (utviklere, ledere, kunden?)
    e. andre grunner?
26. Hvordan synes du det fungerer å bruke metodikken/praksisen på denne måten?
    a. Har det eller forekommer det ofte avvik i bruk av metodikk?
    b. I så fall: hvorfor det?
    c. Er det noe ved denne metodikken/praksisen som du synes burde vært gjort annerledes?
27. Andre metodikker og/eller praksiser, smidige eller ikke-smidige? (hvis ja, gå til *)

Videre spørsmål om metodikk og praksiser:

28. Når ble valgene rundt metodikken og underliggende praksiser tatt? (Tidlig, sent eller fortløpende)
29. Fasilitering: Har du noe inntrykk av hvordan det er for andre å forholde seg til metodikken og underliggende praksiser i prosjektet?
    a. Vanskelig for noen? er dette et problem?
    b. Hvordan sørger dere for å fasilitere bruk av metodikk og praksiser? (e.g. Scrum Master)
    c. Hvis ikke; hvorfor gjør dere ikke dette?
30. Min oppgave fokuserer på dette med å bygge eller tilpasse smidig metodikk…:
    a. Til hvilken grad vil du si at dere er bevisste på dette med å *velge, bygge og tilpasse* en egen prosjekt-spesifikk metodikk? (les: en bevisst prosess?)
        i. Hvorfor er eventuelt dette viktig for dere?

Avslutning (5 minutter):

---

31. Er det tatt høyde for fasene:
    a. Funksjonalitet og kravspesifikasjon
    b. Beslutning av løsning og arkitektur
    c. Oppgavefordeling og fremdriftsplan
    d. Utvikling og koordinering av dette (stand-ups, kanban, mer?)
    e. Overlevering til testing og testing/tuning (eventuelt pilotgruppe?)
    f. Overlevering til produksjon og produksjonssetting
    g. Overvåkning og forvaltning
32. Kort oppsummering av viktigste funn (hvis tid)

---

33. Andre ting vi ikke har diskutert som kan være relevant rundt metodikk og metodetilpasning i prosjektet?
34. *Til prosjektleder: tror du det vil være mulig for meg å prate med flere i prosjektet i november?*
35. *Beskriv hvordan forskningen går videre:*
    a. Ville du hatt mulighet til å stille igjen? (ikke nødvendigvis like lenge)
    b. Spør om jeg skal sende det transkriberte materialet til deg for vurdering dersom ønskelig (mtp. anonymitet, korrekthet osv.)
36. Noen spørsmål?
37. Takk for deltakelsen

Intervjuguide - Runde 2

Presentere meg (ca. 2 min)

- Siv.ing. student, NTNU - Datateknologi og prosjektledelse
- Kort om masteroppgaven
    - Valg av metode og underliggendepraksiser og tilpasning av disse (mao. endringer)
    - Dette prosjektet var interessant fordi…
    - Dataen benyttes som utgangspunkt for et casestudie i oppgaven

Praktisk om intervju (ca. 1 min)

- Tar lydopptak, transkriberes, og alle navn på personer, bedrifter, kunde, osv anonymiseres.
- Svar hvordan du vil, ingen svar er feil.
    - Spørsmål på litt åpent format
    - Noen mer spesifikke oppfølgingsspørsmål fra sist

Introduksjon (3 min)

- Hva er din rolle og dine ansvarsoppgaver i prosjektet?
- Hvor lenge har du jobbet i prosjektet?
- Hva er din erfaring med å jobbe smidig tidligere?

Metodetilpasning og kriterier (20 min):

- Jobber dere etter en navngitt prosjektmodell?
    - Hvis ikke: Hvordan klarer dere da å samles om en arbeidsprosess eller metode?
- Hvordan er prosjektmetoden kommunisert?
- Kan du da beskrive den smidige metoden dere jobber etter i dette prosjektet?
    - Hvilken smidig prosjektmetode tror du dere ligger nærmest?
    - Hvordan kom dere frem til, og satt sammen, denne metoden?
- Kan du beskrive hvilke prosjektfaser metoden deres inneholder?
    - Hvordan (metodemessig) planlegger og designer dere løsninger, når går dere da over på utvikling, og hvilke faser følger da etter dette?

- Hvilke av disse praksisene bruker dere i de forskjellige fasene? (skjema)
- Hva fungerer bra og hva fungerer ikke så bra synes du?
- Kan du påpeke noen endringer (mao. tilpasninger) som har blitt gjort i de ulike praksisene (måten dere jobber på)? (det kan være endringer i praksiser, metoden, fasene)
- Hvorfor gjør dere (i så fall) disse tilpasningene?
    - Er det tilpasninger noe dere burde gjort mer eller mindre av?
- Har dere gjennom prosjektet innført nye praksiser eller gjort større endringer i metoden, eller foregår tilpasninger på praksisnivå?
- Hvem foreslår eller innfører tiltakene for disse tilpasningene, og når tas de opp?

    - Hvilket nivå blir tilpasningene gjort? Er det programnivå, prosjektnivå, eller team nivå?
- Hvor ofte blir det gjort tilpasninger i måten dere jobber på?
- Hva er det som ligger til grunn for at dere kan jobbe så smidig som dere gjør?
- Kommer du på andre praksiser som dere bruker?
    - Har disse vært endret (tilpasset) underveis?

- ((Hvordan vil du kort beskrive deres tilnærming til metodetilpasning?))

- Hvordan vil du betrakte kompetansen omkring prosjektmetode, praksiser og tilpasninger til prosjektdeltakerne?
    - Hvordan driver dere kompetanseheving rundt metodikk?
    - Hvordan driver dere kompetanseheving for kunden rundt metodikk?
- Synes du ellers at dere lykkes med den metoden dere jobber etter?

Oppfølgingsspørsmål (5 min):
- Autonome team og roller:
    - Sitter teamene nå kun innad i kanalene, eller jobber teamene på tvers av kanaler? Og hva er forskjellen på ordene kanal, tema og team?
    - Holder dere på å endre dette?
    - Hvilke roller er det i teamene?
        - arkitekt? eller er dere utviklere alle? hvor mange?
        - rådgivere? hvem er det? hvor mange?
        - utviklere
        - designer
        - interaksjonsdesignere
        - prosjektleder
        - forretningsutviklere
    - Hvor mange team er dere nå og hvor sitter de?
- Standups - er det for hele kanaler, eller enkeltteam?
- Har dere større leveranser som dere skal levere innen en viss dato?
    - Hvordan får dere da til continuous deployment for større leveranser (mye avhengigheter osv)?
    - Kort - hvordan foregår prosessen hvor dere bryter ned i små oppgaver?
- Har dere noen måte dere måler gevinstrealisering på i prosjektet?
    - Ja: så det er mye fokus på verdi?
- Kunne jeg fått et screenshot av et trello-brett, eller skrive ned navnet på kolonnene? Eksempel i oppgaven, men fjerne alle detaljer

Avslutning:
- Andre ting vi ikke har diskutert som kan være relevant for metode og metodetilpasning i prosjektet?
- Beskrive hvordan forskningen går videre
- Noen spørsmål?
- Takk for deltakelsen

A.2 Appendix B: Agile Practices Survey

The agile practices survey follow on the next page.

Praksiser	**Veldig lite**	**Litt**	**En del**	**Mye**	**Veldig mye**
Prioritized work list					
Iteration/sprint planning					
Daily stand-up meetings					
Unit testing					
Release planning					
Active customer participation					
Self-organizing teams					
Frequent and incremental delivery of working software					
Automated builds					
Continuous integration					
Test-driven development (TDD)					
Retrospectives					
Burn-down charts					
Pair programming					
Refactoring					
Collective code ownership					

A.3 Appendix C: Confidentiality Agreement

The confidentiality agreement follow on the next page.

STANDARD TEMPLATE

for confidentiality agreements between a student and a company/external organization concerning work on a master's thesis/project assignment (academic work) in cooperation with a company/external organization, cf. Clause 5 in the standard agreement on academic work in cooperation with a company/external organization.
This template is by order of NTNU's Rector 29 August 2011.

AGREEMENT
between

Student at NTNU:	Date of birth: dd-mm-yy

Company/external organization:

concerning confidentiality.

1. The student is to carry out work in cooperation with a company/external organization that is part of his/her course of study at NTNU.

2. The student undertakes to maintain secrecy regarding what he/she learns about technical equipment, procedures as well as operational and business matters that for competitive reasons have importance for the company/external organization. It is the responsibility of the company/external organization to make it absolutely clear what this information includes.

3. The student is obliged to maintain secrecy about this for 5 years from the date he/she completed work at the organization, see the standard agreement for academic work in cooperation with a company/external organization, Clause 1.

4. The confidentiality requirement does not apply to information that:
a) was in the public domain when it was received
b) was lawfully received from a third party without any agreement concerning secrecy
c) was developed by the student independently of information received
d) the parties are obliged to provide in accordance with law or regulations or by order of a public authority.

....

place, date (dd-mm-yy) student

....

place, date (dd-mm-yy) for company/organization

signed and stamped