Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Mads Olsen Nekkøy

Exploring How Developers Work with Data to Understand User Needs: A Case Study

Master's thesis in Computer Science
Supervisor: Torgeir Dingsøyr
Co-supervisor: Tor Sporsem
June 2023

NTNU
Norwegian University of
Science and Technology

Mads Olsen Nekkøy

Exploring How Developers Work with Data to Understand User Needs: A Case Study

Master's thesis in Computer Science
Supervisor: Torgeir Dingsøyr
Co-supervisor: Tor Sporsem
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4900 - Computer Science, Master's Thesis

Exploring How Developers Work with Data to Understand User Needs: A Case Study

Author:

Mads Olsen Nekkøy

June 2023

Abstract

In today's market, with increasing competition, companies need to change and adapt their methods in order to remain competitive. It's a struggle not only to gain more market share but also to retain existing customers. For developers to succeed, it's necessary to better understand their users and their needs, in order to develop software that is aligned with user´s needs. There are several approaches to gaining this understanding, such as interacting with users directly or collecting data while they use the solution. One of the new approaches to software development is Continuous Software Engineering (CSE), which emphasizes the continuous development of software. This approach facilitates the collection of user data continuously to align the solution to user needs based on real user data. To derive the most benefit from such a method, it's important to understand how developers work with data and comprehend the advantages and challenges that come with this approach.

To address these questions, a multi-case case study was conducted in the spring semester of 2023. A total of 10 developers and designers from four cases were interviewed, and a qualitative analysis was performed on the interview data to gain deep insights into how developers work to understand their users.

The results demonstrate that developers believe in the method and its potential value beyond the requirements work, but there are challenges in selecting and analyzing the data to make reliable conclusions. The results also highlight the significant value of involving users, even in cases where user involvement is currently limited. The implications of this project are that developers can gain a better understanding of how to work with data in relation to their users and gain insights into the advantages and challenges that may arise. For researchers, this provides a foundation for further research in the field, which can help mitigate the challenges and enhance the benefits of the method.

Sammendrag

I dagens marked med stadig økende konkurranse er selskaper nødt til å endre metodene sine for å være konkurransedyktige. Det er ikke bare en kamp om å få større markedsandel, men å beholde sin eksisterende andel. For at utviklere skal lykkes, er det nødvendig å forstå brukerens behov. Det finnes flere tilnærminger for å oppnå denne innsikten, for eksempel ved å snakke direkte med brukere eller samle data når de bruker løsningen. En ny tilnærming i utviklingsarbeidet er Continuous Software Engineering (CSE), hvor fokuset ligger på kontinuerlig utvikling av programvare. Denne tilnærmingen muliggjør kontinuerlig innsamling av brukerdata og å anvende dataene for å tilpasse løsningen etter brukernes behov. For å oppnå maksimalt utbytte av en slik metode, er det viktig å forstå hvordan utviklere arbeider med data og å forstå fordelene og utfordringene knyttet til denne tilnærmingen.

For å svare på disse spørsmålene ble det gjennomført en multi-case studie i vårsemesteret 2023. Ti utviklere og designere fra fire ulike caser ble intervjuet. Disse intervjuene ble analysert ved bruk av en kvalitativ data-analyse for å få innsikt i hvordan utviklere arbeider med å forstå sine brukere.

Resultatene viser at utviklere har stor tro på metoden og at den potensielt kan gi verdi utover det å kravarbeid, men det er også flere utfordringer knyttet til metoden. En av utfordringene er å velge riktig data og utføre analyser som gir pålitelige svar. Resultatene viser også at brukerinvolvering har stor verdi, selv i de tilfellene der brukere ikke er aktivt involvert. Implikasjonene for dette prosjektet er at utviklere kan oppnå en bedre forståelse av hvordan man arbeider med data mot brukerne sine, samt få innsikt i fordelene og utfordringene. For forskere gir dette et grunnlag for videre forskning på området, som potensielt kan bidra til å redusere utfordringene og øke fordelene med metoden.

Preface

This is my master's thesis in the subject TDT4900 - *Computer Science, Master's Thesis* at *Norwegian University of Science and Technology*(NTNU). I began my studies with a bachelor's degree in Computer Engineering from NTNU in Gjøvik before deciding to pursue a master's degree in Computer Technology specializing in software systems. Throughout my studies, I have explored various interesting courses. In particular, the course IMT2243 - *Software Engineering* from my bachelor's program in Gjøvik. It sparked an interest in how developers discover, implement, and prioritize requirements. Additionally, I have taken courses in design of user-centred design, which provided me with a glimpse into how designers work with users to create insight used to develop software. These courses cultivated my curiosity about how developers collaborate with users and aligned well with the problem area of this thesis.

Acknowledgements

I would like to express my sincere gratitude to my two amazing supervisors, Tor and Torgeir. Torgeir Dingsøyr has been my advisor at NTNU throughout the pre-study and master's thesis. We had weekly meetings where I could present my progress and receive feedback. It has been immensely helpful, and I truly appreciate the approachability and willingness to address any questions or concerns. I would also like to thank my advisor at SINTEF, Tor Sporsem. He has been instrumental in the pre-study and master's thesis, assisting me in connecting with interview participants, discussing challenges, and overall, helping me stay motivated.

I would also like to extend my gratitude to all those who participated in the interviews. Your contributions have been greatly appreciated. I would also like to thank my contacts who helped me find interview participants; your assistance has been invaluable.

Lastly, I would like to thank everyone who assisted me by reviewing my work, acting as test subjects before the interviews, and engaging in discussions on potential solutions. Your support is greatly appreciated.

Mads Olsen Nekkøy

Trondheim, June 9, 2023

Table of Contents

List of Figures

List of Tables

Terms and Abbreviations

CSE Continuous Software Engineering.

MVF Minimal Viable Feature.

XP Extreme Programming.

1 Introduction

This section will begin by providing an overview of the thesis background and motivation, followed by the research question and paradigm. Lastly, the thesis scope, limitations, contribution, target audience, and structure will be introduced.

1.1 Background and motivation

The software industry is subject to an intense innovation battle, where companies must continuously evolve their way of working to maintain or improve their position in the market (Bosch, 2014). This innovation encompasses many aspects of developing software, including business, technology such as automated test or monitoring tools, or new processes to better complement new business models or technology. The transformation to rapid development allows developers to create software which is more aligned with actual user needs. Additional critical errors and downtime can be corrected more quickly (Fitzgerald and Stol, 2017).

In the early 2000s, there was a shift from plan-driven development to agile software development. Traditional plan-driven development methods were characterized by clearly defined phases of development and long intervals between software releases. The change towards agile software development facilitated a much shorter time between software releases and enabled developers to react more quickly to changes. Iterative methodologies such as scrum or extreme programming release new versions to the users with intervals of between 2-4 weeks (Sommerville, 2015, p. 74).

Additionally, leading to reduced costs and software more aligned with users' needs. More recently, the introduction of Continuous Software Engineering (CSE) put focus on continuous development and deployment (Dittrich et al., 2018). This allows software developers to react even faster to changes in the market and create solutions that are accurately aligned with customer needs (Bosch, 2014). This means rapid changes directly to the users. User involvement is another core concept of continuous software engineering (Johanssen et al., 2018). During the fall of 2022, a preliminary project was conducted to discover the benefits and challenges of user involvement in requirements engineering. The resulting literature review showcased many essential advantages of user involvement concerning software development. The study revealed several benefits, including that user involvement is critical to system success, higher quality requirements, and better requirements prioritization (Nekkøy, 2022).

Yaman et al.(2016) looked at user involvement regarding continuous delivery and found that customer input conformed well with continuous delivery activities. The article mentioned instant feedback, continuous learning, shorter time to market, and improved satisfaction as perceived benefits. However, these changes to delivering software have accentuated the need to be able to collect and analyze data quickly. To make correct and relevant changes for the users, developers' teams need to be able to cope with this need for speed. In addition, Johanssen et al.2019 states that there is a sparsity in the number of reports in the industry on how user feedback is collected and processed.

To understand how developers cope with this increased pressure and realize the benefits of continuous software engineering, it's necessary to gain insight into how developers work with it. Development teams must handle this new type of "flow", or it may lead to more problems than solutions. For example, the process of collecting and analyzing data needs to be completed on time, or it could lead to lower quality of software and reduced usefulness of the system to users (Johanssen et al., 2018, p. 46). Adopting DevOps practices and culture is crucial in facilitating continuous development and rapidly deploying changes to users.

There are also challenges regarding user involvement in general, like identifying the relevant users, communication between developers and users, and keeping users motivated throughout development(Nekkøy, 2022). The problem is multifaceted, and to deal with these problems, there is a need to explore how developers collect and analyze user data in the context of continuous software engineering. The type of data they use and how they select, remove and filter data. There is also a need to explore the rationale behind these decisions.

Insight into how developers work with data will identify challenges and opportunities to evolve and improve the area of continuous software engineering. More insight will enable researchers to better understand the problems, which will help improve data quality. This will lead to more accurate and relevant data, increasing the overall quality and usefulness of the system. This research can also help identify areas which are lacking and need to be further researched. The insight captured in this thesis does not only apply to the context of requirements engineering but could affect other parts of software development, like testing and bug-fixing.

1.2 Research question

This thesis's primary goal is to better understand how practitioners work with user data in the context of continuous software engineering. Similar research has been done before. Johanssen et al.(2019) have researched how practitioners capture and utilize user feedback during continuous software engineering. The article focused on providing information to improve user feedback capture and utilization for requirements engineering. They conclude that user feedback about CSE and requirements engineering has fallen behind. van Oordt and Guzman(2021) tries to gain a better understanding of how new technology is used to automatically analyze user feedback in practice. They conclude that developers are generally aware of tools, but do not utilize them. van Oordt and Guzman(2021) also researched how practitioners collected, analyzed and utilized user feedback during software evolution. The main differentiating factor with this thesis is the focus on gaining a deeper understanding of how developers work with data, while the other articles focus on how they capture and utilize. This thesis will also explore the benefits and challenges of collecting user data.

The research questions are as follows:

> **RQ 1**: How do developers work with user data to create and evaluate requirements in the context of continuous software engineering?
>
> **RQ 2**: What are the benefits and challenges of collecting user data in the context of continuous software engineering?

1.2.1 Research Paradigm

The research paradigm which suits this type of project is positivism. This thesis aims to gain insight and knowledge about how developers collect and utilize user data in a specific context. This conforms well with positivism studies whose goals are to "discover" the world through observations (Oates, 2006, p. 286). The goal is to create a richer understanding of a unique phenomenon rather than explain phenomenons through hypothesis and quantitative data, which are some of the characteristics of positivism (Oates, 2006, p. 286).

1.3 Thesis scope and limitations

This thesis will focus on interviewing developers from four different projects. Having multiple cases will aid in making more generalizable conclusions compared to a single project. Additionally, it can give more insight into the results, as more cases allow for a comparative analysis which could provide more information about how different factors influence the results.

1.4 Thesis contribution

The result of this thesis will give more insight into how developers work with data collection and analysis using continuous software engineering activities and processes. Readers will hopefully gain more knowledge about what data interests developers and, perhaps equally important, what data is not important. The result will also provide a better understanding of the rationale behind developers' choices when analysing data. Lastly, the thesis will provide general insight into how developers work using continuous software engineering, such as user involvement, tools, roles, methods etc.

1.5 Target audience

This thesis is part of the course *TDT4900 - Computer Science, master thesis* and is the final course for the study program *Computer Science* at NTNU(Norwegian University of Science and Technology).

The target audience is researchers and practitioners interested in the topic of *software engineering*. Readers are assumed to have some knowledge of basic theory within software development. Chapter 2 will provide more detailed information about the relevant theory and will aid in understanding the results.

1.6 Thesis structure

Table 1 shows the structure of the thesis with a short description of each section. The thesis is based on the *IMRAD*-model[1].

Section	**Description**
1. Introduction	This section will provide a general introduction about the thesis, including background and motivation, scope and limitations, contribution and target audience.
2. Theory	This section will provide general information about the core software concepts relevant to this thesis. This includes plan-, iterative, and continuous software development. Additionally, a brief introduction to some specific development methods.
3. Method	The methods section will explain in detail the case study and how interviews were used to collect data, also some of the benefits and challenges with the method. Threats to validity such as construct, internal, external, and reliability will also be discussed and evaluated.
4. Result	This section will present the results from the interviews. Providing a detailed overview of the findings, such as approaches for improving the software, challenges with development, and how they are working with data.
5. Discussion	The discussion will interpret the key finds and discuss the result's significance. This includes discussing the use of user involvement and data-driven approaches, the utilization of CSE for different project contexts, and looking at the future for data-driven development. The limitations of the study will also be presented.
6. Conclusion	The conclusion will summarize, synthesize, and showcase the key findings presented and discussed in section 4 and 5. Additionally, implications for practitioners and researchers. Lastly, future work will be presented.

Table 1: Overview of the thesis structure

[1] https://i.ntnu.no/academic-writing/imrad-structure

2 Theory

This section will present the most essential concepts to better understand the topic at hand and aid in better understanding the discussion. First, section 2.1 will introduce different software processes from a historical aspect, starting with plan-driven development, then introducing agile in section 2.2, and lastly continuous software engineering in section 2.3. Each section will contain an introduction about the development process and some of the methods that are associated with the method before showcasing how requirements are dealt with.

2.1 Plan-driven software development

Plan-driven development is a process where the different activities of software development are planned before starting development. This model was first introduced in 1970 and derived from engineering process models used in large military systems (Sommerville, 2015, p. 47). This approach lets companies get an overview of how resources should be allocated for the period ahead. Planning benefits large companies that must plan long-term for investment, staffing, and business development. This approach demands that the customers know what type of product they want and that the requirements are clear and complete from the beginning. This method has the issue of being challenging to react to changing requirements. In reality, customers often don't know what they want, and their requirements may change during development (Prenner et al., 2021, p. 3).

2.1.1 Waterfall

The most prominent plan-driven software development process is the waterfall model (Prenner et al., 2021, p. 3). The method emphasizes extensive planning and documentation associated with plan-driven development. The model contains five stages that reflect development activities, where in theory, each stage is completed before the next start. These five stages represent the fundamental software development activities (Sommerville, 2015, p. 47), they are:

1. **Requirements analysis and definition**: The system's requirements, goals, and constraints are discovered and defined in detail.
2. **System and software design**: This phase outlines the blueprint to meet the requirements. Hardware and system requirements are specified, in addition to the system architecture.
3. **Implementation and unit testing**: The software design is developed as a set of program units. These units are tested separately to ensure that they meet the specification.

4. **Integration and system testing**: The program units are integrated and tested as a complete system to ensure the system meet the requirements defined in the first phase.

5. **Operation and maintenance**: In this phase, the software operates with the customer. Errors, bugs, or requirements that were not discovered are either fixed or implemented. This involves going back to one of the former stages, depending on the type of change.

2.1.2 Traditional requirements engineering

Requirements are descriptions of the services a system should provide and the constraints required for the system. The process of discovering, analysing, documenting, and validating these requirements is called requirements engineering (Sommerville, 2015, p. 102). Sommerville separates requirements twofold: *user requirements* and *system requirements*. This is mainly to differentiate the granularity of the requirements, where user requirements are more abstract high-level requirements. They are statements about the services and constraints a system should provide the user. System requirements on the other hand, are more detailed. They give a detailed description of the systems functions, services, and constraints (Sommerville, 2015, p. 102).

Traditional requirements engineering is related to plan-driven methodologies, where the initial development phase is focused solely on requirements engineering. Figure 3 shows the waterfall method, where requirements engineering is within the analysis phase, and the other software development phases are firmly separated. The requirements process is divided into three main activities (Sommerville, 2015, p. 111), which are:

1. The initial part of the requirements engineering process is about discovering and analysing requirements. Stakeholders are involved in discovering and defining the requirements for the system. Elicitation, which is the process of extracting requirements, are often done by interviewing or observing relevant stakeholders (Sommerville, 2015, p. 115).

2. The requirements are then put in a standard form. There are many ways this can be done, but they can be either natural language, structural natural language, graphical notations, or mathematical specifications (Sommerville, 2015, p. 121).

3. The final stage is validation. This process tries to ensure the requirements align with what the customer wants. During the validation check, different checks are included: validity, consistency, completeness, realism, and verifiability (Sommerville, 2015, p. 129). In addition, other techniques are used in the validation process; these are: *requirements reviews*, *prototyping*, and *test-case generation* (Sommerville, 2015, p. 130). Validation is an important

stage of requirements engineering, as changes in requirements are cheaper and more easily implemented early in development (Beck, 1999).

The rigidness and focus on planning and documentation make plan-driven development unsuitable for systems where requirements change often. However, large-scale projects and safety-critical systems often demand extensive planning and documentation and might, therefore more suited for plan-driven development (Prenner et al., 2021, p. 3). However, projects where requirements change quickly, are not suited for this type of software development (Sommerville, 2015, p. 49).

Alhazmi and Huang(2020) did a literature where they researched the differences between traditional and agile requirements engineering. They identified five challenges with traditional requirements engineering that agile requirements engineering mitigates, they are as follows:

1. **Communication gaps** - Lack of necessary information leads to unclear visions and roles.
2. **Over-scoping** - Changes that arise during the project and communication gaps can result in the project scope exceeding the available resources.
3. **Requirements validation** - Requirements are continuously changing, and the step-based approach makes it challenging to create requirements that are aligned with users.
4. **Requirements documentation** - This is a resource-intensive activity and does not always provide reliable or precise content.
5. **Customer involvement** - Customers are not heavily involved and typically test the product after development, resulting in additional costs and a product that is less aligned with the customer's needs.

2.2 Agile software development

> "Virtually all software products and apps are now developed using an agile approach." (Sommerville, 2015, p. 77)

Agile software development emerged from dissatisfaction with plan-driven development in the 1990s (Sommerville, 2015, p. 75). Small and medium-sized organisations were frustrated with large amounts of overhead regarding planning and documentation. As a result, there was a wish to reduce the overhead, shifting the focus to software development. The values and principles that form agile are presented in the agile manifesto[2]. Additionally, the plan-driven approach had difficulties reacting to changes during development, something the agile approach embraced (Prenner et al., 2021).

Agile methods develop software incrementally, enabling organisations to react quickly to change. Customer involvement is one of the core principles of agile methods (Sommerville, 2015, p. 76). This means, in practice, that the customer has a more active role in the development and is involved throughout the entire development process (Schön et al., 2017. Agile software development spans the entire software development lifecycle and has created many methods to support this. Methods like Scrum or Kanban are frameworks for organising projects which utilise agile approaches (Sommerville, 2015, p. 85), while other methods like Extreme Programming(XP) focus on practices (Prenner et al., 2021, p. 3). The differences between the plan, iterative, and extreme programming are illustrated in figure 3. The figure shows how the different phases are executed and delivered to the customer throughout the development cycle.

Agile software development facilitates a focus on people rather than processes (Sommerville, 2015, p. 76). This means that developers are encouraged to use the methods and processes that work best for them. Scrum, Kanban, and Extreme Programming are central development methodologies within agile software development, and methods from different approaches are often combined. Therefore it´s important to understand how they work.

2.2.1 Scrum

Agile methods are widely accepted, and scrum is the most popular agile software development method (Hron and Obwegeser, 2022). Scrum is a framework to support agile values and principles. This means that Scrum does not state any techniques for software development but rather how teams should work together to be able to quickly adapt to new changes (Abrahamsson et al., 2002).

[2] https://agilemanifesto.org/

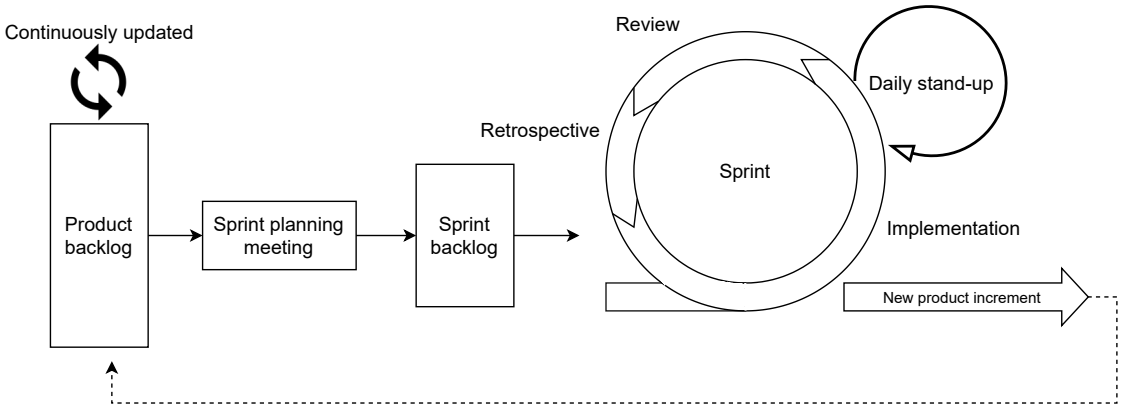


Figure 1: Scrum sprints. Adapted from Agile Software Development Methods: Review and Analysis(p.34) by P.Abrahamsson et al., 2002, VTT Technical Research Centre of Finland. Copyright © VTT 2002

The rationale for scrum is that the development process is unpredictable and complex. Therefore, flexibility is needed in the development process to respond to changes (Abrahamsson et al., 2002).

The development of software is divided into sprints. Sprints are repeating activities for incremental software delivery that typically last between 2-4 weeks (Hron and Obwegeser, 2022). Sprint planning is at the beginning of each sprint; the developers decide which requirements to implement and how it should be accomplished within the time frame.

The requirements are chosen from the product backlog. The product backlog contains a list of everything required for the final product, such as features, bug fixes, or technology upgrades (Abrahamsson et al., 2002). The list is continuously updated and prioritized, involving other stakeholders such as customers, customer support, and management. Next, the requirements chosen for each sprint are put in the sprint backlog; this is nearly identical to the product backlog, but the list is created specifically for a specific sprint.

During the sprint, developers conduct daily stand-ups within the team to update and get an overview of what they and the rest of the team are doing. After each sprint, there is a retrospective meeting to discuss the result of the sprint. The reason is to take lessons from the sprint and use that to increase knowledge among developers (Hron and Obwegeser, 2022). There is also a sprint review where the sprint result is shown to the customer, and the customer can give feedback for the next iteration. For each team using Scrum, there is a ScrumMaster. The role of a ScrumMaster is to ensure that the processes of Scrum are followed and aid the team in using Scrum more efficiently (Sommerville, 2015, p. 85). Another vital role of ScrumMaster is to reduce the interference of other parts of the company with developers so that they can maximise the time used on development.

The systematic review by Hron and Obwegeser(2022) found that Scrum is a very flexible method that is transcending its original setting and purpose, going from small, co-located teams to large-scale, distributed teams. The method is used as a platform for adapting agile to many different contexts and parts of scrum are often incorporated in other methods, like Kanban.

2.2.2 Kanban

Kanban is a process tool (Kniberg, 2010). This means that Kanban is a tool that developers can utilize to work more efficiently. The book *Kanban and Scrum - making the most of both* (Kniberg, 2010) summarizes Kanban with three points:

1. Visualize the workflow
2. Limit Work In Progress(WIP)
3. Measure the lead time

Visualize the workflow refers to using a Kanban board to split tasks into smaller increments and visualise them on a board. This board is divided into columns which represent the workflow; figure 2 is an example of a potential workflow. According to Ahmad et al.(2013), this tool provides developers with valuable insights into the software development process by displaying the allocation of tasks among developers, clearly showcasing prioritized tasks, and highlighting bottlenecks.

Limit Work In Progress or WIP is another central concept of Kanban (Ahmad et al., 2013). The concept centres around limiting the number of tasks for each column of the Kanban board. One instance of this can be seen in figure 2 where the red numbers in each column say how many tasks can be in each; here, we can see a max limit of 2 tasks simultaneously for testing. The team sets these limits, which can be changed under development so that it´s possible to find a balance between the demand and throughput of the team (Ahmad et al., 2013). This results in a sustainable pace of development, which in return creates higher quality software, increased team performance, and builds trust between developers and customers (Ahmad et al., 2013).

Measure the lead time or cycle time refers to the average time it takes to complete a task (Kniberg, 2010). This allows teams to optimize the workflow to reduce the cycle time and make it more predictable. Shorter and more predictable cycles lead to faster delivery of software, reduce over-scoping, and facilitate delivering software on time (Ahmad et al., 2013).

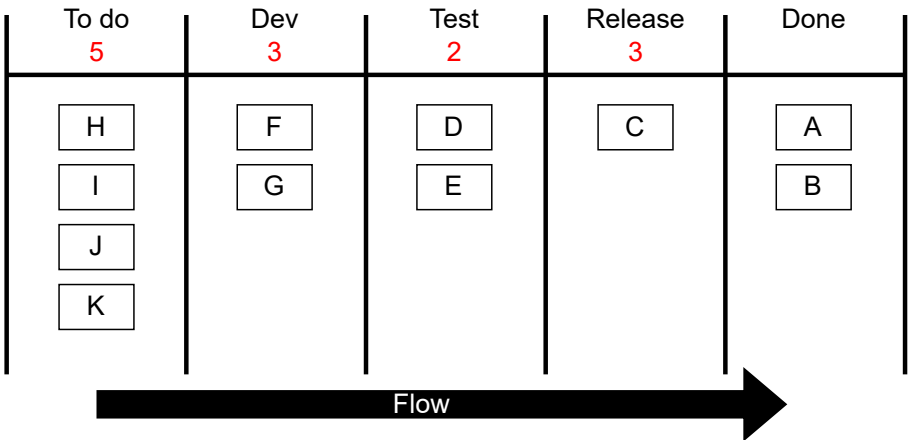


Figure 2: Example of a Kanban board. Adapted from Kanban and Scrum - Making the Most of Both(5), by H.Kniberg, 2010, Lulu.com. Copyright © 2010 Lulu.com.

Kanban is highly adaptive, which allows the process to be easily integrated with existing practices (Kniberg, 2010). An adaptive approach allows more freedom to "do whatever" than prescriptive practices. Prescriptive practices introduce rules and constraints that developers have to follow. Both Scrum and Kanban are highly adaptive. However, Kanban is relatively speaking less prescriptive than Scrum (Ahmad et al., 2013). For example, Scrum incorporates various roles, including the ScrumMaster, and activities such as sprints and sprint-planning meetings. However, these highly adaptive practices have created a challenge because Kanban is hard to use without supporting practices (Ahmad et al., 2013). Kanban can be introduced with agile practices; introducing Kanban with Scrum creates the approach called "ScrumBan" (Ahmad et al., 2013). In practice, the implementation depends on the team and their needs. For example, a team using the Scrum framework could introduce a Kanban board to limit work in progress. Kanban is often combined with other practices, as one of the main challenges reported is that it requires supporting practices (Ahmad et al., 2013).

Ahmad et al.(2013) found that the main motivation for adapting Kanban was simplicity and focusing less on obligatory iterations. They also found many benefits with Kanban, such as customer satisfaction, improved quality and lead time delivery of software, faster feedback, better communication with stakeholders, and increased developer motivation. Kanban was also found to provide organisations with significant value when combined with other agile approaches.

2.2.3 Extreme programming

The term Extreme Programming (XP) was coined by Kent Beck in 1998 (Sommerville, 2015, p. 77). The method is a collection of recognised good practices pushed to the extreme to exploit the reduction in the cost of changing software early in development (Beck, 1999). Instead of planning, analyzing, and designing far ahead as waterfall, these activities are all done in incremental stages throughout development (Beck, 1999). In contrast to scrum, which uses sprints between 2-4 weeks, extreme programming turns these iterative processes to the extreme by having even shorter development cycles or "sprints", as seen in figure 3.

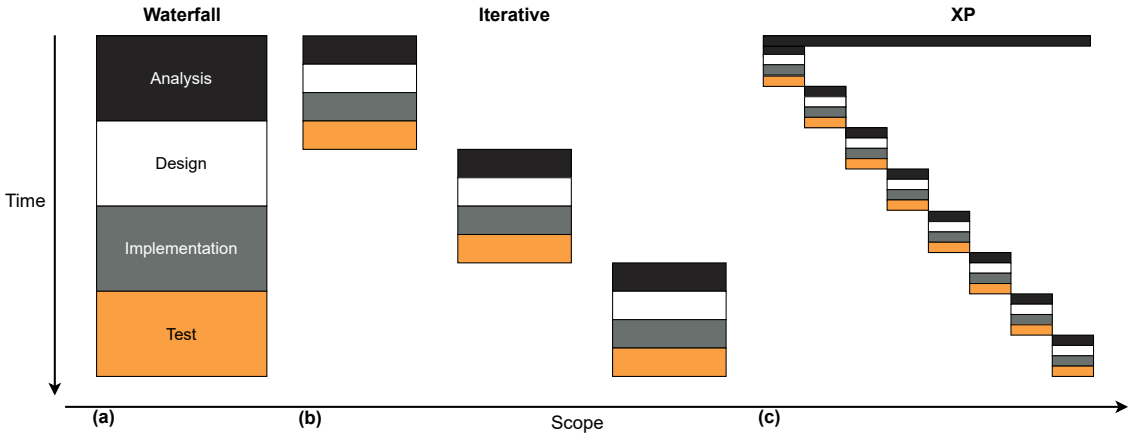


Figure 3: Differences between different development approaches. Adapted from "Embracing change with extreme programming," by K.Beck, 1999, Computer, 32, p. 70. Copyright © 1999 IEEE.

The customer chooses which features that need to be implemented for each iteration. Plan-driven methodologies focus on comprehensive requirements engineering, but extreme programming and agile, in general, integrate requirements elicitation with development by using "user stories" (Sommerville, 2015, p. 79). A user story describes a potential scenario that a system user may encounter. The customer chooses the most valuable stories, and the customer can either pick a set of stories themselves and let the developers calculate a deadline or pick a date and allow the developers to select stories that can be finished before that date (Beck, 1999).

Stories are divided into smaller tasks. Two programmers who work in pairs on a single computer complete the task. This is called pair programming and is one of the core techniques in extreme programming. Developers work test-driven, which means that tests are created before the solution. According to Kent Beck: "if there is a technique at the heart of XP, it is unit testing" (Beck, 1999). Tests are a big part of continuously integrating new code, which is one of the major practices in extreme programming (Beck, 1999). Testing is also a way for the team to build confidence in the

system. There is also a focus on continuously refactoring code as long it´s in scope; big changes that are out of the scope are noted. The customer is available throughout the process to help with questions about scope and implementation approaches.

2.2.4 Agile requirements engineering

Agile requirements engineering differs from plan-driven requirements engineering in that the processes are executed continuously throughout development rather than in a specific part of the development life cycle (Schön et al., 2017). This allows high adaptability to changes in requirements and helps to deal with uncertain requirements (Prenner et al., 2021).

Informal and frequent communication with stakeholders is considered the core of agile requirements engineering, where requirements are elicited, refined, and validated through face-to-face communication (Schön et al., 2017, p. 85). In addition, methods such as surveys, interviews, brainstorming, and focus groups are also utilized for specifying user requirements (Schön et al., 2017, p. 85). These methods are not exclusive to agile requirements engineering but are also applied to traditional requirements engineering. However, the requirement specification document, a core aspect of traditional requirements engineering, is replaced with a product backlog, a list of prioritized requirements (Schön et al., 2017).

User stories are often a part of the product backlog. A user story is a scenario that a system user might experience (Sommerville, 2015, p. 79). These stories are developed in conjunction between the developers and the system user. The system user helps the development team prioritize the requirements which give the most value and can realistically be implemented within an iteration. Each iteration lasts about two weeks (Sommerville, 2015, p. 80). Stories which were not implemented in the original iteration, or implemented stories that are required to be changed can easily be introduced for the next iteration. This is one of the reasons user stories are widely used in agile requirements engineering (Schön et al., 2017). User stories are typically considered more relatable to people compared with requirements documentation, which helps aid in getting users involved with suggesting new requirements (Sommerville, 2015, p. 80). Prototypes are also utilized in agile requirements engineering. They provide multiple benefits, including the elicitation and validation of requirements, as well as being used for requirements documentation. Prototypes also facilitate communication and knowledge sharing between stakeholders and developers (Schön et al., 2017).

As mentioned in sub-section 2.1.2, agile requirements engineering mitigates many of the prominent challenges with traditional requirements engineering. The benefits of agile requirements engineering are summarised by Alhazmi and Huang(2020):

- Lower process overheads
- Better requirements understanding
- Reduced over-scoping
- Responsiveness to requirements changes and prioritisation
- Improved customer relationship

The continuous process of requirements engineering within agile software development conforms well with Continuous Software Engineering (CSE), as the process of discovering, defining, prioritising, changing and implementing new requirements is carried out through development.

2.3 Continuous software engineering

Agile methodologies enabled developers to create software aligned with users' needs and embraced changes throughout the development process. In recent times, there has been a focus on lowering the barrier between development and deployment, allowing changes to be quickly and seamlessly detected and pushed into production. Continuous software engineering(CSE) centres around the continuous development and deployment of software (Dittrich et al., 2018). The approach is characterized as blending development, deployment, and operations. Developing software in short iterations and quickly deploying the changes to users (Johanssen et al., 2018).

Continuous Software Engineering (CSE) is a collection of activities and processes that enable continuous delivery of software (Johanssen et al., 2018). These activities support continuous integration of work, continuous learning, agile practices etc. These practices' effects have enabled Amazon to reportedly deploy to production every 11.6 seconds (Fitzgerald and Stol, 2017). Many of the elements of continuous software engineering have been established for a long time, but the term continuous software engineering is relatively new (Johanssen et al., 2018). This explanation of continuous software engineering is primarily based on the article *Practitioners' Eye on Continuous Software Engineering: An Interview Study* (Johanssen et al., 2018). The article gives an overview of the different definitions of continuous software engineering and presents an overall understanding of the categories and elements that are a part of CSE. The complete Model showcasing the relationship between elements and categories is visualized in figure 4. The article discovered five perspectives from interviews: *tools, method, developer, lifecycle, and product management.*

1. **Tools**: allow for greater automation, leading to a higher development speed. This could, for example, be tools for collecting and analysing data, automated pipelines to facilitate continuous integration and development, or automated tests.

2. **Methodology**: Methodologies that facilitate a workflow where the time from commit to build is as fast and seamless as possible. There is a focus on short iteration and feedback, where changes are quickly presented to the users.

3. **Developer**: Focus on minimising other processes irrelevant to developing software. This is reflected in Scrum, where the Scrum Master´s job is to interface with other parts of the organisation, letting the developers focus on developing software (Sommerville, 2015, p. 85). Another essential characteristic is the focus on creating a safe environment for development and testing.

4. **Life cycle**: The lines between development, deployment, and operational phases are blurred. The time from development to production is short, and the continuous nature allows new functionality to be incorporated throughout the system´s life cycle.

5. **Product management**: The type of product determines whether continuous software engineering can be applied. There is a need for constant funding, new requirements, and a business model applicable to continuous software engineering.

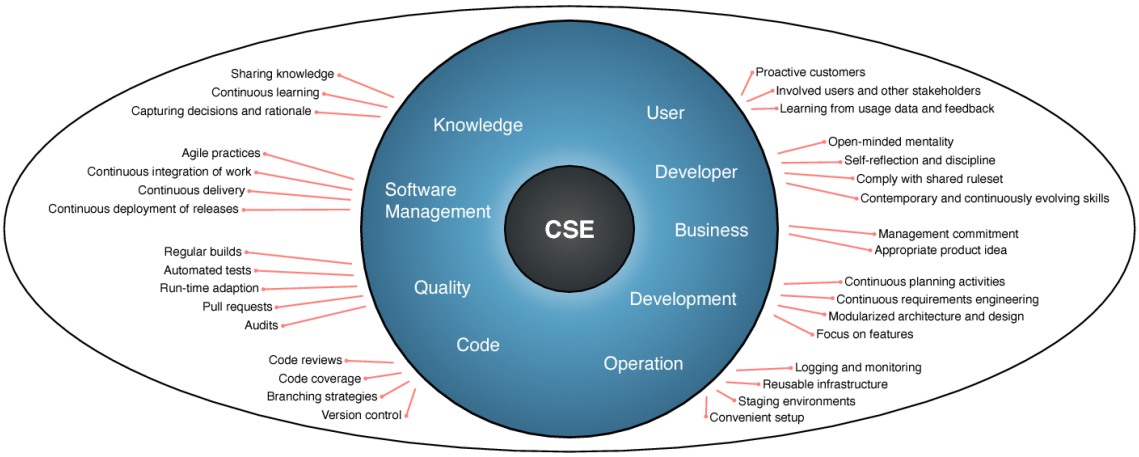


Figure 4: Model of the eye of continuous software engineering. Reprinted from "Practitioners' Eye on Continuous Software Engineering: An Interview Study" by J.O.Johanssen, A.Kleebaum, B.Paech, and B.Brugge, (2018, May 26-27). Practitioners' eye on continuous software engineering. Proceedings of the 2018 International Conference on Software and System Process, Gothenburg, Sweden. https://doi.org/10.1145/3202710.3203150

Agile software development is similar to continuous software engineering; however, CSE looks at software engineering more holistically (Fitzgerald and Stol, 2017). There is a need to look more comprehensive than just continuous software integration, such as introducing agile approaches to

other organisational functions such as finance and human resources (Fitzgerald and Stol, 2017). The article "Continuous software engineering: A roadmap and agenda" by *Brian Fitzgerland* and *Klaas-Jan stol* argues that continuous software engineering is more than continuous delivery and deployment, where the continuous software engineering is seen as a holistic process beyond the implementation of specific methods, to support the entire software production from customer to delivery(Fitzgerald and Stol, 2017).

Continuous experimentation and innovation

An important factor for continuous software engineering is work towards continuous customer experimentation and innovation. The fast cycles and rapid feedback facilitate experimentation and innovation. Development costs are reduced as new features focus on customers' wants, and unwanted features are quickly discarded. Another essential aspect is the ability to quickly collect, analyse customer data, and deploy changes to the customers (Bosch, 2014, p. 25). Developers using this approach often execute experiments using A/B testing. A/B-testing is when there are two different software versions; this could, for example, be two different log-in screens or layouts for a news site. The two different versions(A/B) are tested on separate user groups, where the versions are compared for developers to make informed decisions based on user data (Bosch, 2014, p. 4).

2.3.1 DevOps

DevOps is a software development methodology that bridges the gap between development and IT operations. These elements encompass two(development and operation) of nine elements that define the *eye of CSE*, seen in figure 4. Additionally, it covers many of the elements in the figure, such as *continuous delivery*, *continuous integration of work*, *regular builds*, *automated tests*, and *sharing knowledge*. This makes DevOps a central approach in continuous software engineering.

The introduction of DevOps requires technology-heavy changes, but equally important is the need for culture change (Ebert and Hochstein, 2023). Some of the benefits of this approach are to deliver value fast to the customer, reduce problems caused by miscommunication, and speed up problem resolution (Ebert et al., 2016). Figure 5 illustrates the DevOps process. Note that throughout this continuous development, there is continuous collaboration and communication with all parties involved (Ebert and Hochstein, 2023). The proceeding "What is DevOps?: A Systematic Mapping Study on Definitions and Practices" (Jabbari et al., 2016) defines DevOps as:

> "DevOps is a development methodology aimed at bridging the gap between development(Dev) and Operations(Ops), emphasizing communication and collaboration, continuous integration, quality assurance, and delivery with automated deployment utilizing a set of development practices. (Jabbari et al., 2016, p. 5)"

There is a need for culture change to bridge the gap between developers and operations and realise the benefits of DevOps. Developers, quality assurance, and operations are required to collaborate closely and remove the "siloes" that have been established (Ebert et al., 2016). Cultural change is so significant because the different roles have conflicting motives. Developers wish for change, testers want to minimise risks, and operations want stability. This means that roles need to evolve. For example, developers must expand their knowledge beyond programming and acquire additional skills such as database administration and testing (Ebert et al., 2016).

Automation is one of the core concepts of both continuous software engineering and DevOps (Jabbari et al., 2016, Johanssen et al., 2018). To deliver on both quality and short cycle times, automation is needed (Ebert et al., 2016). Tools are essential to automated processes. These tools can, for example, automate the process of compiling code, managing dependencies, creating documentation, testing, or deploying code to production.

Logging and monitoring tools are also used with DevOps. Logging tools can help resolve performance issues or improve troubleshooting. Monitoring tools enable the organisation to be proactive regarding the health of the system, where problems are identified and ideally resolved before the end-user is affected (Ebert et al., 2016).

DevOps extends agile. The practices of DevOps can be seen as pragmatic extensions of agile activities (Jabbari et al., 2016). For example, the principle of satisfying the customer through early and continuous delivery of software[3] is well aligned with the definition of DevOps.

[3]https://agilemanifesto.org/principles.html

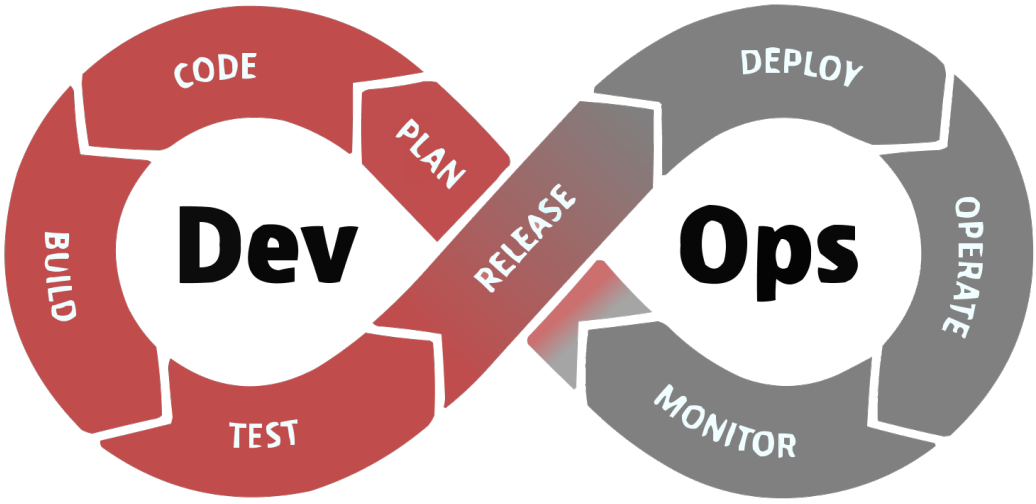


Figure 5: DevOps process. Adapted from "DevOps in Practice," by C.Ebert and L.Hochstein, 2023, IEEE Software, 40, 29-36. Copyright © 2023 IEEE.

2.3.2 The Hypex-model

The Hypex model or Hypothesis Experiment Data-Driven Development model is a development process that facilitates experimenting and aids in shortening the feedback cycle to facilitate continuous software development (Bosch, 2014, p. 160). As with other experiment systems, the typical aspect is that requirements evolve in real-time based on actual user data rather than being rigid and based on opinions from the development team (Bosch, 2014, p. 157). The models facilitate a continuous capture, analysis and implementation of features based on user data, which is key to continuous software engineering.

The Model divides the creation, development, and implementation of features into five steps; these steps are presented in figure 6

1. **Feature Backlog Generation:** The first step concerns discovering features that bring value to the customers. Features are generated based on strategic business goals and the product management and development team´s understanding of their customers. The outcome is a feature backlog with potential features that might be implemented.

2. **Feature selection and Specification:** The second step involves choosing features for experimentation. They mention three approaches for selecting features; the first is to choose features that are believed to have a gap between expected and actual behaviour. The second is where a functionality covers an area with minimal prior experience and the third concerns

areas where there are many approaches for implementing a feature. After the features are selected, the expected behaviour is defined. This means defending how the feature adds value to the customer and supports the business goals. Defining desired behaviour allows for quantitative analysis and creating a discussion about the behaviour.

3. **Implementation and Instrumentation:** This step focuses on implementing the smallest part of the functionality that brings value. Bosch(2014) refers to this as the Minimal Viable Feature (MVF). The feature is developed in iteration, and data is collected for statistical analysis throughout the deployment to discover the actual behaviour.

4. **Gap Analysis:** In this stage, the actual and expected behaviour are compared to determine whether the functionality is sufficiently developed to meet the desired behaviour. There are three outcomes from this stage. First, if the gap is small enough, the feature is finalised. The second alternative is if the gap is substantial, the development teams start creating hypotheses to explain the gap. The last outcome is that the feature is abandoned because the expected benefits are not achieved, and the current implementation indicates no- or negative- effect on actual behaviour.

5. **Hypothesis generation and Selection:** If there is a gap between expected and actual behaviour, the next step is creating a hypothesis to explain the gap. They define two categories for hypothesis: 1) the MVF is not sufficiently developed. This involved further developing the feature and collecting more accurate metrics. 2) an alternative implementation of the MVF will yield a different outcome.

6. **Alternative Implementation:** The last stage is concerned if the outcome of stage 5 is an alternative implementation. The solution is to perform A/B-testing, where data is collected between both versions. Then, the differences can be analysed and used to validate the hypothesis. Finally, when sufficient data is collected, the teams can move to stage 4, gap analysis.

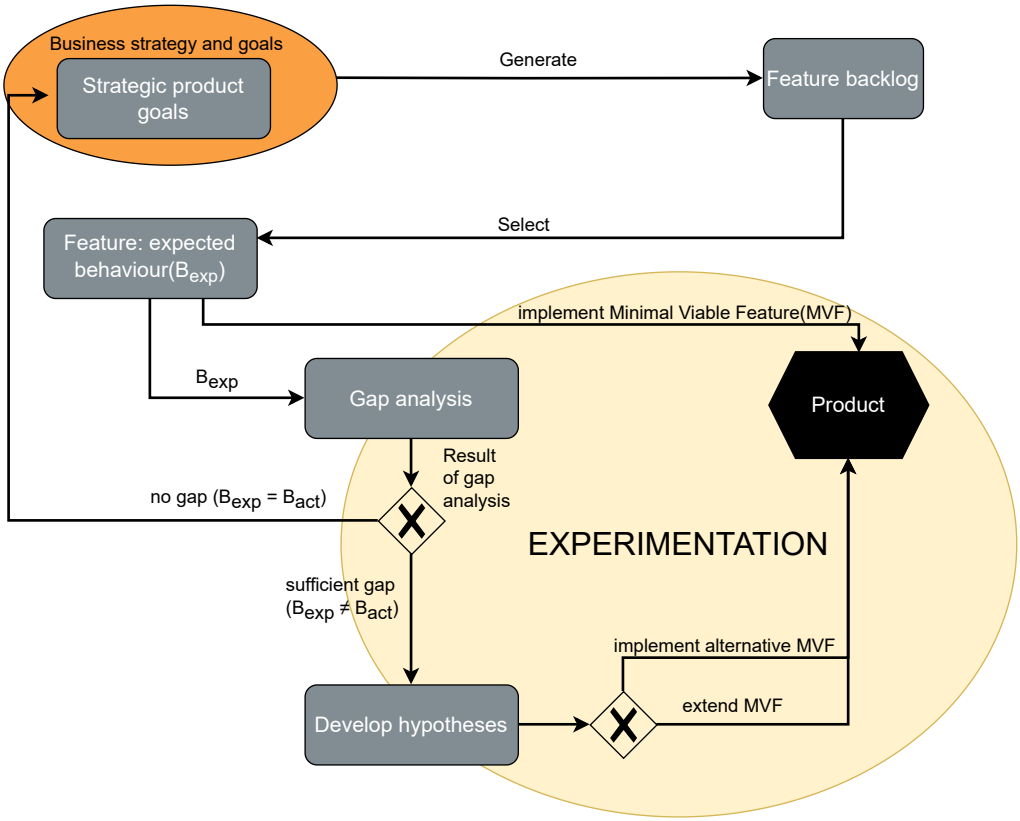


Figure 6: The HYPEX-model. Adapted from *Continuous Software Engineering: An Introduction*(p.160), by J.Bosch and H.H.Olsson, 2014, Springer Champ. © 2014 Springer International Publishing Switzerland.

2.3.3 Continuous requirements engineering

Introducing methods like continuous deployment facilitates the ability for developers to observe and react to changes in the software quickly. This allows developers who have established continuous development and deployment to use instant user feedback to validate functionality (Bosch, 2014, p. 18). This means that the software is continuously aligned with the needs and desires of the users (Niu et al., 2018). The differences between agile and continuous requirements engineering are hard to define since CSE and agile are closely related. However, continuous software engineering incorporates agile practices(see figure 4) and applies these practices to support a continuous workflow. This could allow developers to explicitly test high-risk assumptions on the users, where feedback is collected and acted upon in short feedback cycles (Niu et al., 2018).

3 Method

This section presents the research design and method used to answer the research question from section 1.2, as well as the choices made regarding validity and reliability. Firstly, the research strategy will be described, followed by an explanation of how the data was generated, processed, and analyzed. Lastly, the method used will be evaluated, and any limitations of the study will be discussed.

3.1 Research process

The sequence of activities in a research project or the process is one of the 6Ps of research according to Oates(2006) and is essential for the research to be considered rigorous. Figure 7 is from Oates(2006, p. 33). This figure provides an overview of the research processes for this specific research process. The icons coloured in red illustrate the path that is chosen for this research.

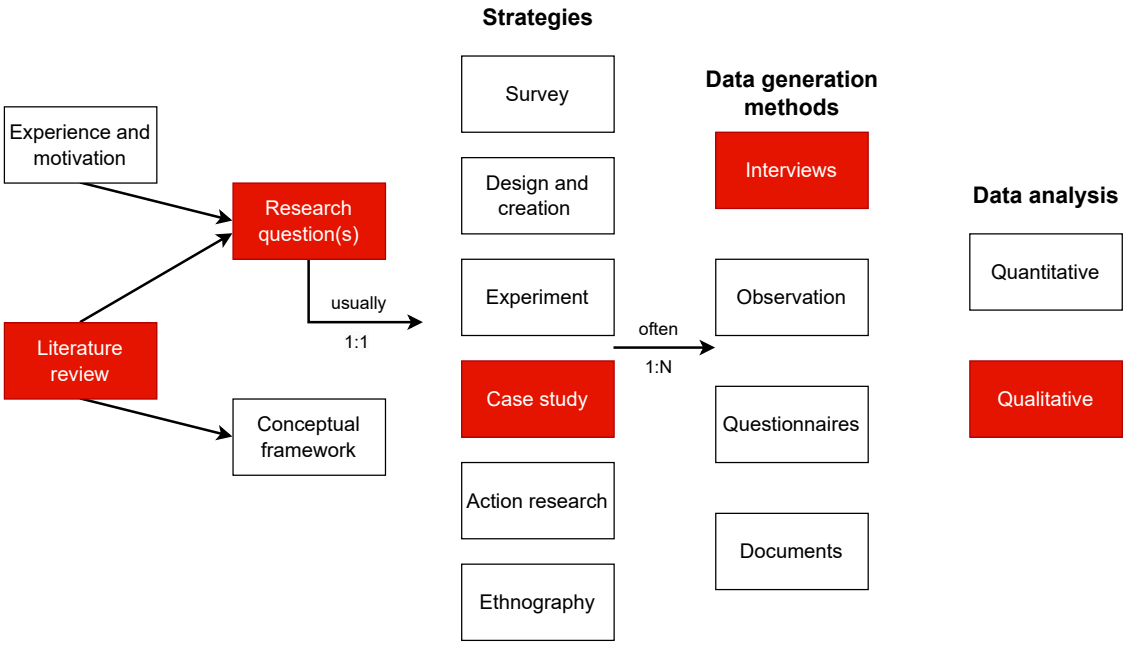


Figure 7: Model of the research process. Adapted from *Researching Information Systems and Computing*(p.33), by B.J.Oates, 2006, Sage Publications Ltd.

3.2 Research strategy

The research strategy chosen for this study is a case study, as mentioned in chapter 1.1. The study aims to gain insights and a deeper understanding of how developers work with data to understand their users and the reasons behind their choices. The strategy and methods are based on the book "Researching Information Systems and Computing" by Oates(2006), which provides the foundation for the choices made.

Case studies are suitable for capturing rich, detailed insights into the phenomenon under investigation (Oates, 2006, p. 141), which in this case is the software development practitioners' use of data in development. The result of such a method is to gather insights and create knowledge that can help advance the research area and apply it to other domains.

The book mentions three types of case studies: exploratory, descriptive, and explanatory. This case study is an exploratory case study which focuses on exploring and uncovering new insights. As continuous software development and data-driven requirements engineering are relatively new and unexplored areas, it aligns well with what is referred to as an exploratory case study. It could be argued for an explanatory case study, which aims to provide a rich and detailed understanding of a specific phenomenon by examining different influencing factors or connecting findings to existing theories to identify similarities or differences. However, an exploratory case study is more appropriate for areas with limited existing theory, where the goal is to understand the phenomenon under investigation better and generate new ideas or hypotheses to be further explored. Additionally, case studies vary in their temporal approach, and in this case, data has been collected about the present, making it a short-term, contemporary study (Oates, 2006, p. 141).

This is a multiple case study, meaning that numerous cases are investigated rather than a single case study. The main argument mentioned by both Yin(2002) and Oates(2006) is that it enhances credibility for readers and makes the method more robust. The drawback is that it requires more time and resources. Additionally, Oates(2006) states two disadvantages of case study research as:

1. Sometimes being perceived as lacking rigour, leading to generalizations with low credibility.
2. Difficult to gain access to the required resources(setting, people, or documents).

3.3 Cases

This section will introduce the different cases, starting with an overview of how the various projects were selected and how the individual interview subjects were contacted. Then, a brief introduction will be provided for each case to give an impression of the context of the cases.

3.3.1 Selecting participants

For Cases A and B, the selection of projects was made with the help of a contact person in the company, where the desired type of project and the number of interviews were communicated. Naturally, these preferences were considered, but time and availability had to be considered when selecting projects. I preferred developers who worked directly with users and preferably had experience from other projects. Unfortunately, the results were in a project context where they did not work directly with customers. However, case A was a well-established project which utilized DevOps practices and a combination of agile methods. Case C provided valuable information about how developers desired to work. However, due to the early phase of development, capturing information pertaining to the research questions proved challenging.

Case C involved a designer working for a large international consulting company. Contact was made through a contact person in the company who referred me to the relevant person, similar to Cases A and B. Case C provided useful insight into the perspective of a consultant and the challenges with both working and accessing users.

Case D was provided to me by the advisor from Sintef, Tor Thorsrud Sporsem. This was because of the difficulty of obtaining enough interviews from Cases A and B. The interviews were highly relevant as they involved designers working with user data directly from their customers and employing methods that are relevant for continuous software engineering. Moreover, the interviews were already transcribed.

For Cases A, B, and C, the procedure for contacting the interview subjects was the same. Through my contacts in the respective companies, I was provided with a list of email addresses of potential candidates. First, an email was sent introducing myself, explaining the purpose of the task, and providing general information about the interview. Additionally, a consent form was attached(see appendix D), detailing the participation process and how privacy and data would be handled. Then, individual emails were sent to schedule interview times, and meetings were set up via Microsoft Teams.

3.3.2 Case A

Case A is a financial product belonging to a Nordic consulting company. They offer financial products to banks and car dealerships, among others. They develop a credit processing system to automate loan applications, including credit checks and document generation for customers who want to take out loans for things like cars or boats. The product is often integrated into an already existing solution. This leads to the development team working towards the customer instead of the end-user. They use a range of methods in development, including sprints, Kanban boards, and DevOps practices.

3.3.3 Case B

Case B is an internal project within the same company as Case A. It is referred to as a data platform with the primary purpose of using data to identify and showcase employees' skills. In addition, it serves as a reference tool that employees in the company can use to see information about projects, competencies, roles, and general data about the employees. This internal project is meant for developers that are currently not on customer projects, which leads to very high throughput of developers on and off the project. The development method is characterised by one developer from the project as having "process debt", which means they do not currently have a very established method for developing the software.

3.3.4 Case C

Case C is a project aimed at providing support and counselling for children and young people up to 18 years old. Through various communication channels such as phone, email, or chat, individuals can communicate with trusted individuals about anything in their lives, both the good and the bad. The project offers a counselling service through a website that emphasises creating a safe environment for children and young people to receive the help they may need. The project utilises methods to work directly with the end users. However, the designer does not work with the end-user directly and gets the result from user participation activities and mainly works towards the project members. The interviewee is a consultant working as a designer with the project team.

3.3.5 Case D

Case D deals with collecting, processing, and sharing data about public transportation. The goal is to create a solution that makes public transportation easy and efficient for both users and providers of public transportation. This includes displaying available transportation options, providing route guidance, and facilitating ticket purchases. They also assist transportation providers by offering a ticket sales system that makes it easier for new operators to enter the market. They utilise several activities in the development process to work directly with the end-users, including interviews and lab testing.

3.4 Data generation

The case study offers various sources of evidence, including documentation, interviews, or observations (Yin, 2002, p. 86). The approach for this study was interviews, which is a commonly used approach in case studies (Oates, 2006). Interviews provide an opportunity to gather detailed information on specific topics (Oates, 2006; Yin, 2002). Additionally, the approach allows for an in-depth exploration of experiences, opinions, and emotions (Oates, 2006). These attributes conform well with the goals of this thesis, as it enables a deep dive into how developers work in order to understand their users and the challenges they encounter.

Observations were considered as they can provide a better understanding of the developer's context by capturing what happens. However, observations were not chosen due to time constraints and limited resources. Observations are resource-intensive and time-consuming (Yin, 2002). Conducting interviews via video conferencing is significantly less time- and resource-consuming. Figure 9 in Appendix A presents how the data was collected and categorized.

3.4.1 Secondary data

Secondary data is data that is collected for other purposes, and often by another researcher (Jacobsen, 2015, p. 140). The other type of data is primary data. This refers to the data that is collected directly from participants for the first time. The interviews from cases A, B, and C are primary data and as mentioned in sub-section 3.3.1 the data from case D was collected from another research, thereby making it secondary data. The secondary data typically focuses on a different problem area and aims to address different research questions, which may result in the data being less relevant to one's own problem area. Fortunately, secondary data was relatively aligned with this project, making it highly valuable.

3.4.2 Interviews

Oates(2006) defines three types of interviews: Structured, Semi-Structured, and Unstructured. The difference lies in the level of control over the interview, where a Structured interview follows a set of predetermined questions, while an Unstructured interview is more like a conversation around a topic without any pre-defined questions. The choice was a Semi-Structured approach. This is because it allows for specific themes common to all questions but also provides flexibility to add and explore even further. According to Oates(2006), a semi-structured interview is suitable when the research aims to explore, which aligns with the objective of this study.

People respond differently based on how they perceive the interviewer (Oates, 2006), so it was necessary to be perceived as a serious actor. Specific considerations included responding politely to emails and during interviews, being punctual, and removing any headwear during interviews. Another perspective was to avoid being perceived as threatening so that the interviewee would be more open to discussing topics rather than becoming defensive. Yin(2002) mentions that in a context where deeper exploration is desired, using "how" questions instead of "why" questions can prevent interviewees from becoming defensive. Striking a balance between being "friendly" and "non-threatening" while delving into the topics had to be considered during the interviews.

Before conducting the interviews, two rounds of practice interviews were carried out. One test interview was born with a classmate, while the other was with a developer. They provided feedback on formulating questions, order, and any aspects they thought should be included. They also gave me some experience in conducting interviews, and especially in becoming more comfortable and better at asking follow-up questions, resulting in more in-depth answers. It was important to ask open-ended questions as they generally elicit more detailed responses (Oates, 2006).

The interviews were conducted via video using Microsoft Teams. Conducting the interviews via video offered flexibility with regard to finding available times, as the only requirements were an internet connection and a laptop or phone. Physical interviews would be more challenging as the interviewees and researcher were in different geographical locations. The interviews were recorded for later reference, and transcription was done using the built-in transcription feature in Microsoft Teams. This allowed for automatically transcribing interviews. Conducting interviews had numerous advantages, it was not without its challenges. de Villiers et al.(2022) reported some of these challenges as:

- Silence in video calls creates anxiety
- Delay in sending and receiving a message creating a perception of the responder being less friendly or focused
- Feeling of being watched, thus creating stress

The execution of the interviews was based on Oates(2006)'s explanation of conducting interviews. The interviews began with some small talk to make the interviewees more comfortable, followed by introducing the purpose of the study and providing information on their rights. The rights consisted of a summary of the information letter sent to each participant via email. This included asking permission to record, explaining who has access to the recording, the possibility to withdraw at any point, and how the data would be processed and deleted. After this, some simple "warm-up questions" were asked, such as "Tell me about yourself?" before exploring more relevant topics. The main method to mitigate challenges with interviews via video was making the informants as comfortable as possible and appear interested, friendly, and focused. For example by mentioning things they said earlier to appear focused, apologising if I interrupted, and showing enthusiasm.

3.5 Coding and Data analysis

After the interviews and transcription were finished, they were anonymized. This involved removing the names of interviewees, project names, company names and other information that could identify the interviewee or the company. Moreover, the transcription was imported to NVivo, an analysis tool for qualitative research. NVivo was used as the primary tool for data analysis; it allowed for collecting and analysing all files, which included creating categories to organise the data, searching between all files, and making notes directly in the files.

3.5.1 Qualitative data analysis

The qualitative analysis revolves around reducing large, unmanageable amounts of data to smaller manageable parts to combine the pieces and identify relationships, differences, patter, contexts or underlying causes (Jacobsen, 2015). Different approaches to qualitative analysis exist, such as text analysis, conversation analysis, process analysis or content analysis. This project utilises a content-analysis approach, which assumes that collected data can be fragmented into smaller components and systematically categorized together (Jacobsen, 2015). Text analysis is a well-established method used in research on a master level (Jacobsen, 2015), it also conforms well with the goal of the thesis which is to gain insight into how developers work with data and understand

their reasoning, rather than explaining the processes involved. The book *Hvordan Gjennomføre Undersøkelser?: Innføring I samfunnsvitenskapelig metode* (Jacobsen, 2015, p. 199) defines four condition in order to execute a qualitative analysis in practice, these are:

1. **Document:** Collect and organize all produced data. In practice, this was transcribing interviews and importing them to NVivo. The process of transcribing interviews is something that was extremely underestimated in the beginning, even with help from tools for transcribing, the process of correcting text was comprehensive and time-consuming.

2. **Explore:** Go through the data un-systematically and search for emerging themes or interesting findings. The finding was noted in a notebook and NVivo as comments for later use. The notes seemed trivial at first, but writing notes underway in this phase was extremely valuable for later, as these "trivial"-thoughts were actually of great value.

3. **Systematize and categorize**: To create order from the large amount of overwhelming information, the findings are put into categories set by the research and have some commonalities. This allowed for all findings in the transcribed material to be gathered in one place, enabling the identification of similarities and differences across the cases.

4. **Connect:** This part is about seeing similarities and connections between the categories. For example, some categories are related to a broader theme or connected to a similar situation. Figure 8 illustrates how the codes were connected to themes in this analysis. Gathering multiple codes together created more a holistic view of the problem area. One important lesson from this stage was to not be afraid of making changes, work iteratively with the themes and ask others for their opinion.

Jacobsen(2015) describes two different approaches for creating categories/codes: *first-cycle coding* and *second-cycle coding*. The first approach involves taking root in the data that is to be analysed and creating codes based on data that are relevant to each other. Some codes were created pre-analysis, but based primarily on the interview guide from appendix C and the introduction from section 1. The other type involves creating codes based on the existing codes created during the *first-cycle*. This could be new codes or higher-level codes.

For this project, the *first-cycle codes* defines the categories/codes, whilst the *second-cycle codes* defines higher-level codes or themes, as seen in figure 8. At this point, the data is split into smaller and more manageable pieces of information, and the work to identify relationships, differences, patterns, contexts, or underlying causes can begin.

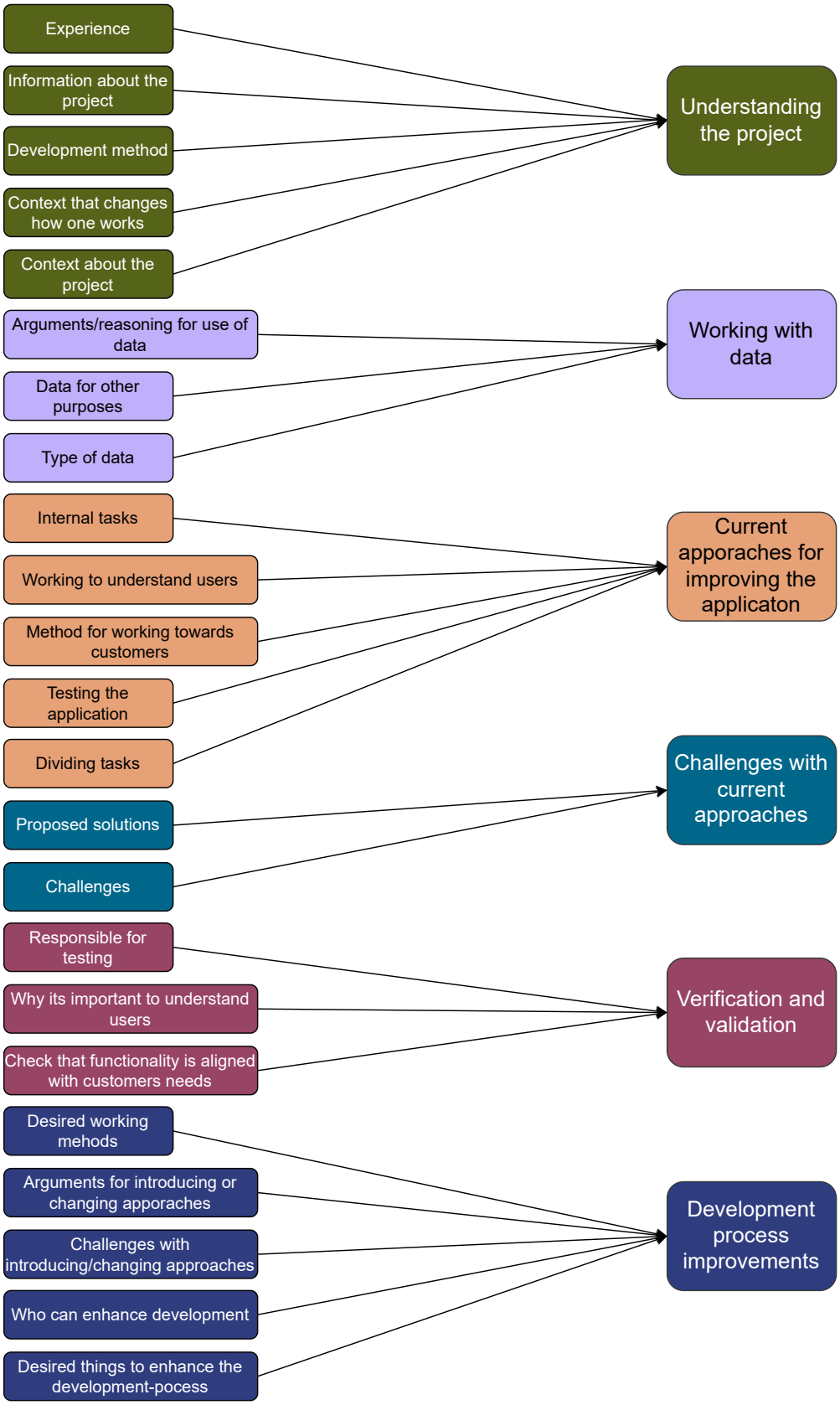


Figure 8: Discovering themes from codes

3.6 Study validation

This sub-section will explain the different forms of validation, such as construct, internal, external and reliability. These forms of validation will be presented, in addition to showcasing how they affected the projected and how they were mitigated. Lastly, a final evaluation of the general quality of the study will be presented.

3.6.1 Construct validity

Construct validity refers to which degree the research investigates what it claims to investigate, in other words, how accurate the research is to an observation of reality (Gibbert et al., 2008). Yin(2002) defines three tactics to increase construct validity: *multiple sources of evidence*, *chain of evidence*, and *draft case study report to be reviews by key informants.*

Multiple sources of evidence are supported by the research strategy, which is a multiple case study. Four different cases allow for triangulation of data from various sources. There was a discussion of adding observations as a data generation method. However, due to limited time and resources, this was not possible. Observations would have increased the construct validity even further.

Additionally, the choice of cases and interviewees was limited and mainly controlled by what was available. This is not optimal, as the selected cases were not necessarily aligned with the problem area initially stated. This could cause a misalignment between what the researcher was supposed to investigate and the actual investigation.

> "Every case should serve a specific purpose within the overall scope of inquiry."(Yin, 2002, p. 47)

The chain of evidence is the second tactic to increase the construct validity. The methods used to collect data and the interview guide used for the interviews are provided in appendix C. Additionally, the results are thoroughly presented.

The last tactic is to present the draft case study report to key informants to ensure that the report is aligned with the informants. There was initially planned to do this activity, but mainly due to a lack of time caused by getting access to informants at a late time, the activity fell through. This could be problematic as the researcher's interpretations of the collected data could diverge from the participants' intended meaning.

3.6.2 Internal validity

Internal validity concerns the relations between the reality described by the researcher and the actual reality (Jacobsen, 2015). The issue is whether the research provides good arguments and logical reasons to defend the research conclusions (Gibbert et al., 2008). This type of validity is not applied to exploratory studies such as this and is therefore irrelevant (Yin, 2002, p. 34).

3.6.3 External validity

External validity refers to which degree the findings can be generalised to other contexts (Jacobsen, 2015, p. 237). Jacobsen, 2015 defines two types of generalization, *generalising to other units in the same case* and *generalisation to other cases*. Both types of generalisation are dependent on two conditions. They are *number of units* and *how the units are chosen*.

The more data that is collected, the more likely the findings will be generalised. A total of 10 interviews were spread between 4 cases, where 2(A and B) cases achieved saturation. This is when newer interviews stop providing new and interesting information and indicates an adequate participant number (Jacobsen, 2015). However, there is a shortcoming in only having 1 and 2 interviews for cases C and D, respectively. This reduced the external validity, as the data collected was inadequate.

With regards to selecting units, there are three types defined by Jacobsen(2015): *The typical selection*, *Spread*, and *Least likely units*. For this project, the selection approach can be argued as a spread. As presented in sub-section 3.3, the cases are vastly different concerning the customer and the general context. This strengthens the external validity.

Another aspect that might affect the external validity is the interviews in case D. These interviews are over one year old and are, therefore, not the most up-to-date information. Thus, the opinions and information from these results may have changed drastically or be invalid. In conclusion, I believe that the interviews provide valuable information that is relevant. However, age should be taken into consideration, as the early phase of testing new methods makes their opinions subject to change.

3.6.4 Reliability

Reliability is concerned with reducing the errors and biases in a study (Yin, 2002, p. 37). To minimise these aspects, there are several conditions to look at. The first look if the data generation method has affected the participants. Study participants will be exposed to different stimuli and signals (Jacobsen, 2015).

One aspect is how the researcher is perceived and interacts with the people involved. This is mentioned in sub-section 3.4 where it is explained how the research tried to be perceived by the interviewees regarding the choice of clothes, behaviour, and how questions were formulated.

The second aspect concerns the context in which the data is captured. People act differently depending on different contexts, so a natural setting where people are comfortable is preferred (Jacobsen, 2015). The interviews for this project were performed using Microsoft Teams, which meant that all interviews were conducted via video. This could potentially lead to differentiating results contra having the interviews physically face-to-face. Video conference tools offer many benefits, but also some challenges as mentioned in sub-section 3.4.

How one was perceived and the context of video conferencing may have influenced the results somewhat. I believe the results would have been different if they had been conducted by a researcher who had done it face-to-face instead of over video because it feels more "real" or "serious." From my experience, it is easier to delve deeper into things when physically present. As mentioned by Oates(2006), finding a place where the interviewees are comfortable is essential. This is reflected in Appendix C, where one can see that the interview lengths are somewhat shorter than the planned 45 minutes. Many reasons could cause this, but there are perhaps two main reasons for this. The first is that the researcher is not very experienced with interviews, and the second is that the interviews were done via video, which was personally not the most comfortable place to conduct interviews.

Another aspect of reliability is related to collecting and analysing data. Tools assisted in collecting data, automatic tools did the transcribing, and the interviews were recorded for later use. Therefore, aiding in reducing the likelihood of errors in the recorded data. Additionally, the transcription was read-through whilst listening to the recording to increase the transcribed material's accuracy.

The final aspect is related to analysis. Although a crucial part of the analysis is categorising and creating themes from the data, it is something not to be taken lightly. The codes and themes developed in this stage are the baselines for the findings, and the categorisation is not always straightforward. These issues were mitigated by iterating over the codes and the themes. However, more could be done to increase the validity. In retrospect, the codes and themes should have been inspected and discussed with other research.

3.7 Evaluation

The quality evaluation is based on a list of questions presented in Jacobsen(2015, p. 247). How the data was collected, processed, and analysed is well documented. Additionally, the methods used are described in detail, showcasing how the method was selected to align with the problem area and how other methods were evaluated. This results in a project which can be re-created. Criteria for choosing specific cases and interviewees are also described, reflected, and argumented for.

There are some forms of validation in this project, the main form of validation being the data triangulation using multiple cases to interview a wide range of participants. The weaknesses with validation are fully transparent; they have been discussed and acknowledged. As this is an exploratory case study, connecting the findings to existing theory is challenging, and some findings stand out from the research. However, some findings also align well with existing theories and research.

The researcher has relatively limited knowledge about the field of study. However, the pre-study before this project increased my knowledge and insights about the field of study. The divide between results and the researcher's interpretations is clear, but it is clear that the interpretation is based on the results and other related research in the field. Lastly, how different aspects of the study could have affected has thoroughly been discussed and reflected upon. To conclude, there are some weaknesses in this thesis. These might have affected the results, but overall I believe that the quality is sufficient for a thesis on a master's level.

4 Results

The results are divided into five different chapters: Approaches to improving the software application, Challenges with current development processes, Working with data, Validation of the application, Desired changes to the development process, and lastly a brief summary. Table 2 shows each case's different roles and participants. The chapter is structured to show the result for each case.

Case A		
ID	**Role**	**Date**
P1	DevOps/Developer	23.03.2023
P2	Developer	22.03.2023
P3	System architect/Developer	29.03.2023
P4	Developer	27.04.2023
Case B		
ID	**Role**	**Date**
P5	Developer	29.03.2023
P6	Developer	13.04.2023
P7	Developer	17.04.2023
Case C		
ID	**Role**	**Date**
P8	Designer	27.04.2023
Case D		
ID	**Role**	**Date**
P9	Designer	14.01.2022
P10	Designer	23.03.2022

Table 2: Overview of interviews and role for all four cases

4.1 Approaches improving the software application

In case A, the development can be seen as two-fold, where one part focuses on adapting and developing the software for each customer. The second part focuses on improving the overall application, regardless of the application. The customers have direct contact with the teams that are associated with them, where they can come up with requests for new functionality or report on issues. These requests are sometimes supplied with justification for change by the customers, where new regulations for loan applications or user feedback could cause changes. These requests are handled by the team lead, who will present them in a meeting to discuss implementation. The discussion deals with what must be done to implement the changes and how long it will take. The issues are then posted on Trello and approved by the customer, and then they start implementing.

A developer from case A states:

> "It's actually a relatively extensive process, or rather it varies a bit from team to team, but for many of the teams, it's a slightly more involved process."

Teams may use different tools depending on the customers' preferences. These tools are Trello, Jira, GitHub Issues, or other help desks. The teams also share functionality with other customers. Another developer said:

> "For example, if a customer comes up with a request and we create a solution for it, it often happens that we realize other customers have similar needs or requirements. We can then go and say that we have created a solution for this."

Additionally, they have monthly internal meetings to discuss application changes. These changes are not specifically for a customer but for the entire application. An example of such a change is updating a cloud service's latest version. These changes can also provide new functionality to the customers without them explicitly asking for it. An example could be that updating the cloud service provides access to new third-party applications or introducing updated data encryption.

The software that Case A develops is often integrated into already existing solutions. This means the design must suit the customer and follow their existing design language. This work is outsourced to a designer, who gets information in the form of a design profile from which the designer can then work out and adapt the design to the customer. The designer then creates mock-ups for the development team, which are discussed internally before the final design is implemented.

"We receive five different designs that he has created, where he tries to make them as different as he wants or feels like. Based on these examples we receive, we have internal meetings to gather feedback from people in the department and to decide what we want to go for technically and in terms of design. First, we discuss which of these five designs looks the best and then we go through the technical details. Is it feasible? Based on these two factors, we choose the one that most people think looks the best and is feasible."

Case B is an internal project for employees not currently on customer projects. New functionality or issues come from both the team- or tech- lead and the developers themselves. One developer states:

"It's a work in progress at the moment. So far, it's been the team lead and tech lead who identify something they would like, and then we implement it."

Another developer states:

"Historically, it has actually only been a backlog filled with things to do that has been continuously updated as those working on the project have discovered errors or deficiencies."

It is mentioned that the project has newly been given product owners who are in the process of creating insight. They have been creating questionnaires and interviewing users to understand their needs.

In case C, the project´s primary users are kids and young adults between 7 and 18 years old. The project offers kids a safe place to talk with someone by mail, phone or chat. The focus is on creating a safe and anonymous place to talk. The interviewee is a consultant who works with a team from the client, where the consultant works with the user experience. The client utilizes user testing; they go to schools that fit their age group and have different activities, such as: interactive prototypes created in Figma or posters with a range of different versions of an interface.

"And they have also had a lot of[user testing], they have brought out a poster where they have drawn a lot of different sketches, and then they ask you to point out what you[target group] like best, so it's very observation-based."

The consultant then receives the data from the user testing, creating the basis for further developing the solution. After changes are made, the solution is then presented to the client, which in return gives feedback before they test it on users.

> "If you have something that is already developed and the service starts to take shape, it is usually that you then go to the customer and present what you have and get feedback. So it's often almost like a mini user test, but it happens with your customer contacts instead of the larger user group. That's where we get most of our feedback."

This feedback is then used for assigning new tasks. There are two designers on the team, and they used this feedback to split into smaller tasks and assign the tasks based on how much time they have. There is a focus on time because they only have a certain amount of hours each week for this specific project.

> "It's mainly about running an iterative process with the customer, starting with a discovery meeting. We develop some sketches, and then we sit down together. They go through the sketches, identifying everything that needs to be changed in the next round."

The client meeting is mainly walking through the changes made, where the designers ask for specific details. For example, if the title is correct, the colours are fitting, or questions about the size of a particular button.

Case D works with an application that aims to help people travel using public transport. This includes showing routes, live-tracking of transportation, and ticket solutions. The interviewees are similar to case C also designers. They have started working with data analysis and hypothesis-driven. They collect and analyse data to observe the behaviour of their users. They begin by making a hypothesis. One designer states:

> "Yes, often we get feedback from the customer and you hear things, feedback from people in the organization or sometimes it's just a gut feeling, and then you make a hypothesis that, for example, with Vipps login, by using Vipps login more users might log in who were planning to do an anonymous purchase, or maybe even create a profile with Vipps so they don't have to fill out anything, they just log in. Then we get the data and say okay."

The hypothesis is created, and then the next step is to find which data is necessary to confirm or disapprove the hypothesis. One example could be gathering the number of users who click on a button and comparing it with how many used to click the button before a new feature or re-design was introduced. The results could then say that they observe a 50% increase in users clicking that button, where the results help decide the next step. However, collecting the "correct" data is not an easy task.

> "So one just has to make a bit of a hairy guess as to what is a good indicator, I'm not that good even though I have statistics and at the university, I'm not that good at statistical significance and such things. So yes, we often see it more as an indicator than an exact answer."

Lastly, they decide if the hypothesis has been confirmed or disproved. The result determines if they implement the changes or not. Additionally to the hypothesis and data-driven approach, they also utilize something they have termed "adoptive testing". They describe it as a form of user testing performed on people who are easy to reach, such as friends, colleagues, or people on the street. The reasoning behind this type of testing is to have a low threshold for user testing. They use adoptive testing and hypothesis-driven development to get a "hunch" on whether the hypothesis could be of value.

The respective cases operate in very different contexts. Case A developing software that is sold as a financial offering to banks and retailers, focusing on automating loan applications and credit checks, among other features. They have multiple teams dedicated to managing and working with their respective clients.

Case B is an internal project that works on various projects based on an employee information data platform. The project is developed by consultants who are in between client assignments and generally work on the project for short periods of time.

In case C, a consultant working on a project aimed at providing a safe space for children and youth to communicate with adults was interviewed. The project team conducts the activities to understand the users, while the consultant receives the results from these activities and uses them to validate the changes.

Case D revolves around designers who employ a variety of user-centred activities to make navigating public transportation in Norway as easy and comprehensible as possible. They have a strong focus on their users and perform several activities to ensure that the changes are made with consideration for the users. Unlike cases A and C, they work directly with end users.

4.2 Challenges with current development processes

Although developers are not directly involved in user testing in case C, it is mentioned that conducting user testing with children and young people is a challenge. According to the developers, the challenge lies in understanding how children think. Surprising feedback is often received, such as children being afraid that the solution will be too cute and wimpy, some choices of words being difficult to understand, or some illustrations being interpreted differently.

One developer states:

> "Often in the dialogue with the client, they put focus on clear language, structure, and focus on what is happening during development. But for children, it's mostly about the overall impression of how they experience the solution. To make it work for them, it's important that they get that good feeling, which is impossible to develop without being involved."

In this project, it is mentioned that the current client is good at user testing. It adds tremendous value, but it is noted that it can be challenging for other projects to conduct user involvement, as they charge hourly, and the client generally also has existing design and development tasks. These challenges show that it can be both difficult to work with certain users depending on the context, and difficulties in gaining access to users.

Hypothesis and data-driven development are mentioned as a method that developers have tremendous confidence in for case D, where they can access actual user data and observe how users use the solution in practice. This enables them to check changes against actual user data. However, certain challenges are mentioned, one related to hypotheses about the data that needs to be collected to test the hypothesis.

A designer mentions:

> "But again, the difficult thing about data is that it's hard to know if you're interpreting the data correctly, if you're getting the right data, if the data is comprehensive and complete, and those kinds of things."

The interviews are conducted with designers, and challenges with using analysis tools are also mentioned in that context. One of the designers says that a developer with technical competence must help set up the correct code parameters and transfer them into the analysis tool to track changes.

The designers also mention:

> "I find it incredibly challenging[set up tests in the application], and I usually can't do it unless I have a developer to help me. So yes, it's a difficult field, and you have to be pretty aware of your own limitations, I think, when working with data."

Another challenge is related to the process of testing hypotheses, and one of the designers mentions that: "You have to really think about what data you need to confirm or refute your hypothesis." Data analysis requires knowledge, and it is essential to have good competence to avoid making hasty decisions based on incorrect information. This illustrates that even though one has access to and the opportunity to utilize user data, there are challenges in terms of how to make the best possible use of it.

Case B is recognized for having a high rate of developer turnover due to being a project for developers between customer projects. This leads to some challenges with development. The developers and the team- or tech-lead update features and new addition to the backlog. The high throughput of different developers of varying skill levels creates challenges with describing items in the backlog.

A developer states:

> "It varies a lot from developer to developer how good they are at describing issues, and some just think, "Now I'll make it real quick, and then I'll fix it later," and maybe they forget. So then you end up with a task with a title longer than the description of the task itself."

Additionally to the description of tasks sometimes being difficult for developers to understand, there are issues with the tasks that are left in the backlog for longer. This leads to the backlog being filled with old tasks, which might not be relevant any longer and can cause problems with the system. The project has issues with the high throughput of developers and a lack of permanent resources, which can help mitigate some of the issues. The interviewees from Case B mentions a lack of process and setup for the development. One developer says "process-debt" when discussing how to improve the current processes. These challenges indicate that there is a need to have defined methods and processes in order to successfully develop software, and that project requires permanent resources on the project.

Case A has minimal interaction with the end-users and receives information about new features or issues from their customers. This leads to the development team losing some of the "control, " making them dependent on the customer. This is not necessarily a challenge, but one developer mentions when talking about using tools for data collection:

> "You might miss the unforeseen events, for example, if a customer gets stuck in the process because they did something strange that I haven't tested, so it's difficult for us to identify."

Another issue is that the external APIs sometimes make it difficult for the developers to work with the API. A developer mentions it´s a lot of "stabbing in the dark and seeing what data comes through". This highlights the importance of good documentation, as it makes the code more difficult and frustrating to work with. The other issue is related to not collecting data about the users, illustrating one potential benefit of collecting data that can be applied to testing new features.

In summary, the findings revealed a variety of challenges. These challenges encompassed difficulties with users and access, inadequacies in the existing development methods and processes hindering successful software development, and missed opportunities arising from not utilized activities that could bring significant benefits for developers.

4.3 Working with data

Case A utilized mock data from different sources. Mock data is fabricated data that is used to mimic real data. Services like Altinn, The Norwegian Public Roads Administration (NPRA), Tenor or The Norwegian Tax Administration provide them with mock data. The reliance on mock data results from the type of product they create, namely a product for the finance sector. For example, data about a user's owned vehicle and tax returns are needed for a loan application. The mock data allows the developers to simulate the "flow" of the system to ensure that the correct data is shown, allowing them to use fabricated personal information such as name, birth, income, tax etc. For one specific feature, they might not need all the data that they have access to; one developer mentions:

> "We also review the data, and here we can see how many cars are registered to the customer, for example. Then, we can suggest to the customer that we can actually display information on how many cars they have and how much mortgage is left on the car, and things like that. So, we look at the data and come up with new proposals for the customer based on that."

Besides proposing new features based on data from different sources when working on customer features, they do not collect data for analysis. However, they mention that some of the customers use Google Analytics to gather information about the users. This could, for example, be how long a loan application takes per user or which step users spend the most time on. This analysis might then be used to justify changes.

In case B, the project is based on a data lake that gathers employee information from their employee systems. Simply explained a data lake is a place to store and process large amounts of data. Different systems are then created on top of that, for example, a type of application that works like a yellow-pages for employees. Besides working with or generating data for the different applications, the development team does not systematically utilize data to discover new features or identify issues. However, some developers mention that with the introduction of product owners, they have started collecting data to create insight. One developer mentions:

> "I think they have started the initial round internally with questionnaires. They have interviewed me, and I guess they have also interviewed others, so it's a mix of qualitative and quantitative methods."

For case C, the data is mainly gathered from user testing done by the client. The type of data is mainly feedback that is captured from user participant activities that are used to improve the solution. What sets the project apart from "other" projects is that the data collected is not only used to develop the solution. The designer claims the data is used more for the client's purposes than UX and development. The data is used for various purposes such as research and mapping the solution, exploring which groups use the solution, or observing children's behaviour.

A designer mentions:

> "They [the client] focus a lot on anonymization, and how can we best reach out to these children and help them with what they need. So, I think they learn a lot from seeing the design process of that solution and how the children use it."

In case D, users can give written feedback to the developers from the application; the feedback is collected in a Slack channel, allowing them to read and review the feedback they receive quickly. This is one of the channels they use to come up with new hypotheses. The primary data collection happens by analyzing how users interact with the solution, such as collecting data on how many people click a button and measuring change over time. This data is then fed into an analytics tool like Google Analytics, where they evaluate the data collection results.

4.4 Validation of the application

The context of case A where functionality is based on the customer´s request and not the users directly. For the team to validate the functionality, they have a testing phase. A developer states:

> "So, we have a trial run first where the customer who placed the order tests both as an end-user and as a case officer and tests the entire system on their end. When we go into production, changes are usually noticed quickly, so we get notified quickly if there is anything that is not working as they expect."

This process ensures that the functionality is both validated and verified.

For case B, there are some processes for ensuring functionality validation. They have a test environment where they can push changes or new features before production. One developer states:

> "We do have a test environment. I think he[product owner] is asked to go into the test environment and check things continuously. But it's quite easy to do since there is a low threshold for running things, as it's only for internal use."

Additionally, it's mentioned by the developers that many of the potential issues are taken care of by the designers. This could, for example, be that the team-lead wants new functionality, and the designers get feedback and work with the team-lead to create a solution. Next, the developers can use the work from the designers and implement a solution.

One of the developers from Case B mentions the importance of validating that the solution aligns with the users' needs. At a previous employer, they developed a solution without considering how and where the application would be used and ended up creating an application that could not be used.

> "There were forestry workers who were supposed to use a touchpad, but they couldn't use it because of the cold, as they got so cold in their fingers. And then you have created an unusable system because the app they created for the touch screen was supposed to be used only in the field."

In case C, the designers follow an iterative process with the client. As mentioned, they create sketches, engage in dialogue with the client, and make changes based on client feedback. When the client is satisfied, the sketches are implemented. The designer mentions this in the context of how they confirm changes in the solution:

> "Sketch, check with the customer, implement the changes, and then check again."

As mentioned in the method, case D is secondary data and therefore did not follow the most relevant questions or approach. What can be said is that they use the data collected on user behaviour as an indicator to confirm or refute hypotheses. In this way, they see how users respond to changes. For example, they may add support for Vipps to register and then measure an increase in the number of orders for registered or unregistered users.

Cases A and B are somewhat similar in that they push changes in a test environment before they put it into production. However, in case A the process seems more organised process with internal and customer testing, as well as channels to receive feedback when in production. Case C follows an iterative approach where the customer controls when the changes are satisfactory. Case D is entirely different from the rest of the cases, as they collect data directly from the users in order to make decisions on changing or implementing new features.

4.5 Desired changes to the development-process

This chapter is divided into two sub-section, user involvement practices and development methods. The desired changes are presented with the following argumentation of why they can be of value.

4.5.1 User involvement practices

Several developers related to case A mention that they want to work in a user-centred way and focus more on universal design. The sketches outsourced to others in the company are mentioned as specific examples of something they want to work on. The use of observations is also mentioned, with the idea that by using this method, developers will gain insight into how users use the application and capture usage patterns that might differ from what they think. They have the opportunity to collect data and use tools like Google Analytics to gain even more insights, but it is mentioned that they do not feel a particular need for this at the moment, and it is more ”nice to have” rather than ”need to have”.

> ”I think we could really improve the quality of the product by getting more developers to be aware of design and actually lifting the quality of the product quite well. I also believe the technology stack could be updated faster with newer technology instead of sitting on old code.”

One of the developers from Case B mentions that gathering data about the user, such as usage patterns, can be used to make the application safer. The person mentions that they work proactively to create a rigid application. It is said that, for example, gathering usage patterns can be used to test the application so that if a user inputs something outside of the ”normal”, the system will not crash.

Another developer wants to introduce more demos of the solution and a "demo culture", which means that developers are encouraged to show their work to others on the team. It is argued that this will increase product ownership and knowledge among developers, as they can share experiences, give tips, and ask questions to each other.

There is a desire for user involvement and data analysis, wherein in data analysis, one can see where users stop in the application and how much time they spend on a certain point, for example. The argument is that this gathers qualitative and quantitative data based on actual usage data. It will reduce the assumptions developers have to make in development and create more insight where usage patterns that may not have been foreseen can be captured. Another developer mentions something similar with MVP testing, where changes are pushed out to test user interest, where, like data/hypothesis-driven development, data is used to check if functionality interests users. A/B testing is also mentioned as a method to supplement development, where one can check if what is being developed matches what users want.

In Case C, it is mentioned that it would be desirable to work directly with the customer and eliminate the intermediary between the developers and the client. The argument is that it will increase the value of user testing for the designer and create more insight.

For Case D, they want to continue with what they call "adoptive"-testing, which is regular testing on fixed resources. The argument is that it supplements hypothesis-driven development well, as it costs less time and resources than "classic" user testing. It can quickly help designers get a "hunch" that the hypothesis is on target.

Summary

The results indicate that the different cases have different approaches to working and engaging with end-users. In case A, there is minimal contact with end-users, whereas in case D they heavily rely on end-user opinions and behaviour. These distinct contexts also raise different challenges. In some cases there are difficulties reaching the end users. In others, there are issues with the lack of methodology and processes, and some acknowledge the potential loss of not utilizing methodologies.

The cases also differ in their utilization of data. Some use data to simulate functionality, while others use data to analyze users or make changes based on it. Regarding validation, there are similarities between some cases, but, as with the other topics discussed earlier, there are differences among them. One common aspect that can be identified is that all cases aim to work directly with their end-users or are already doing so.

5 Discussion

This section will discuss the results presented in chapter 4 in light of the research area introduced in chapter 1. First, the advantages and challenges of applying a data-driven approach will be discussed, followed by how my initial thoughts deviated from the findings. Since the results are based on four different cases, there will be a discussion on how the contexts of the projects impact the development. Then, we will move on to how data and analysis can be used for various purposes, followed by a brief discussion on the future of data-driven development. Finally, the evaluation and limitations of this research will be presented.

5.1 Using User Involvement and Data-Driven Approaches to Enhance Product Development

The results highlight developers' significant value on user involvement in the development process. All cases exhibit extensive user involvement or express a desire to work closely with users, such as through observations. A particularly intriguing case is Case D, where designers actively engage with users and utilize methods to gain insights, notably hypothesis-driven development using user data. This approach ensures that the solution aligns with user needs and preferences. This is aligned with Bosch(2014), which supports this rationale, emphasizing the importance of testing new functionalities before allocating extensive resources to ensure alignment with customer needs. This master thesis explores the rationale behind collecting data and the decision-making process, shedding light on the advantages of hypothesis-driven approaches supported by data.

However, challenges arise when considering the specific types of user data collected. Although the data is genuine, it becomes difficult to ascertain whether it effectively proves the hypotheses being tested due to multiple factors and the complexity of analyzing statistics accurately. For example, an increase in specific functionality or a reduction in error reports does not necessarily establish a causal relationship with a recently introduced feature. Interviewees note the challenges in interpreting and analyzing data and desire more knowledge in selecting and verifying data for their hypotheses. This is also reflected in Bosch(2014), which acknowledges these challenges under the "Industrial Experiences" section, particularly regarding defining metrics and data collection. Hence, the collected data serves as an indicator rather than a final solution or answer.

Being able to perform such analyses without any uncertainty is challenging, as there is always uncertainty with data (Taraldsen, 2022, p. 3). By educating the relevant team members in the field of statistical analysis, I believe that the method will increase credibility and reduce uncertainty regarding both selection of specific data and the analysis. Taraldsen(2022) mentions three questions which are central to the statistical-field, these are:

1. How to learn from data?
2. What are good data?
3. Which data should be observed?

Learning how to answer these questions will help identify and analyse the correct data and create high-quality analysis. This way, developers can be confident that the result represents the real situation so that changes align with what the users want. This is reflected with the designers in case D, where they specifically mention the need for knowledge about the field. Finding the data that can be used for the hypotheses is challenging, and insufficient knowledge can lead to decisions based on incorrect data.

Similarly, the generation of hypotheses relies on a "gut feeling" mentioned by the interviewees. Implementing automated feedback services to consolidate and prioritize feedback is worth considering, as suggested by Johanssen et al.(2019). The existing solution where written feedback is delivered in a Slack channel is a good way of automatically collecting user feedback. However, the answer could be extended to map particular feedback to specific features or issues. Although not a perfect solution, such a tool would aid in finding and prioritizing the most valuable requirements.

Using what they currently refer to as "adoptive" testing seems an excellent way to gather feedback on the hypotheses they create quickly. By doing this, they can challenge, discuss, or modify the hypotheses. Such conversations can also help provide increased insight, as other individuals often have different perspectives than oneself, thus leading to a more robust hypothesis. One of the prerequisites for it to be a suitable method is that it is quick and cost-effective, allowing them to do it continuously. Involving users in development requires additional resources, and might be challenging for managers to overcome these costs (Abelein and Paech, 2015; Bano et al., 2017). This is also represented in case C where some clients do not want consultants to involve users because it takes a lot of resources. Having a method for involving users that is not as resource-intensive might help decision-makers be more inclined to introduce methods like "adoptive" testing.

The purpose of adopting a data-driven approach, as discussed by both interviewees and the literature (Bosch, 2014, Johanssen et al., 2019), is to bring the product closer to the customer and eliminate decision-making uncertainties. However, for this approach to outperform existing methods, it is essential to base decisions on accurate and precise data. Without proper knowledge of the topic and a lack of understanding of the limitation, the results would be features that are based on incomplete and wrong data and can lead to frustration, decreased user satisfaction, and even potential negative impacts on their work or tasks performed through the application.

5.1.1 Potential privacy concerns

Employing methods that gather user data, often without explicit knowledge, potential privacy issues may arise. Several interviewees mentioned the use of the analytics tool Google Analytics. Recently, Google Analytics has faced scrutiny due to data transfers to the United States, with the Norwegian Data Protection Authority issuing a temporary decision stating that such data transfers conflict with General Data Protection Regulation(GDPR) regulations[4]. This case highlights the security risks and privacy concerns associated with using such services. The collected data can be sensitive, and if it falls into the hands of third parties, it can be misused for identity theft or financial losses.

5.2 Expectations versus Realities: User Involvement, Data Analysis, and Development Methods

Before conducting the interviews, certain assumptions were made regarding the expected findings. Based mainly on educational experience and limited industry exposure, it was initially believed that there would be a comprehensive focus on user participation Methods such as prototyping, interviews, and user testing from user-centred design were expected to be utilized across all cases. This aligned with the preliminary project on *user involvement in requirements engineering*, which emphasized the importance of user involvement for developing systems with high user satisfaction. Most cases confirmed these expectations, particularly Case C, which highlighted the significance and challenges of accessing users, aligning with the preliminary project's findings.

The assumed value of involving users is presented in every case, even when they are not currently involving users in developing new features. The importance is shown in all cases, and table 3 elaborates on this connection.

[4]https://www.datatilsynet.no/aktuelt/aktuelle-nyheter-2023/varsel-om-vedtak-i-google-analytics-saken/

Connection	Explanation	Typical quote
Involving user - increased product quality	There is a consistent assumption of the importance and value of including end-users in development to enhance the quality of the product.	(**Case A**) "I think we could really improve the quality of the product by getting more developers to be aware of design and actually lifting the quality of the product quite well." "So we haven't necessarily had a lot of dialogue as far as I'm aware with, for example, retailers or end customers, which is something I personally would have liked a bit more of."
		(**Case B**) "What I'm a fan of is that we have a user experience team that use methods to gain user-insight so that we have something to work with and can set goals based on that. In other words, it[development] becomes more user-centred." "I would like to have some user involvement."
		(**Case C**)"So in a way, all direct contact with the user is of great value to us. Often much more than what customers can provide, because they have a completely different perspective than the end user."
		(**Case D**)"The advantage, as mentioned earlier, is that you obtain real user data, so you see what people actually do rather than what they say they do. And in that way, I believe it's easier to achieve behavioural changes if that's what you aim for and to create things tailored to how you actually behave." "So all traditional user tests only show what people think they would do in a fictional setting, so it's like- it has its advantages with the qualitative insights you gain from conducting the classic user tests. It can be nice to have occasionally, but in general, I believe that behavioural data provides a better understanding."

Table 3: Elaborating on the relation between involving users and product quality

Surprisingly, Case A revealed that some developers had no direct contact with end-users, relying solely on communication with the client. Although this approach seemed to work well for the project's nature, considering it's been in development for over 20 years, there was a desire to adopt a more user-centred approach and take responsibility for the design internally rather than outsourcing it. However, it did not appear to be an urgent need, contradicting the theory emphasizing the importance of including users for system success (Abelein and Paech, 2015). While they mentioned the ability to analyze user data, it was perceived as a "nice to have" rather than a critical requirement.

Using a user-centred approach can be challenging regarding accessing the end-users, but if achieved, I believe it can enhance user satisfaction. The current process in Case A, where they select from five design mock-ups, introduces assumptions, and it's not guaranteed that they choose the best option or that any of the choices align with users' preferences. By implementing user-centred methods, these assumptions can be reduced, allowing for the introduction of a user interface, in this example, designed based on input from actual system users.

This suggests that developers have some recognition for collecting user data and that it can provide value. With that being said, it is mentioned as something they believe can be valuable, not something they have tested and can say with certainty. They perceive it as a potentially valuable approach for gaining deeper insights into how users genuinely utilize the application, potentially providing a stronger foundation for implementing and validating changes.

5.3 Project context for Continuous Software Engineering

The results are based on interviews from four different cases. Johanssen et al.(2018) discusses the definition of Continuous Software Engineering (CSE) and mentions that context is essential, as the project's nature must align with Continuous Software Engineering for it to work. Not all types of projects can apply this development method. This sub-section will discuss the project context with regard to Continuous Software Engineering, as the research questions are relevant for gaining a deeper understanding of collecting user data in the context of CSE.

This conclusion may initially seem applicable to Case A. The project's nature does not align with some of the perspectives of Continuous Software Engineering as presented in Figure 4, considering that their developers do not have direct contact with end-users, and the design is outsourced. On the other hand, they work continuously in many areas, such as adopting agile practices, receiving continuous customer requirements, and utilizing various DevOps practices crucial for continuous integration and delivery of new iterations. The product they create is also designed for continuous development, as it serves diverse customers with varying needs, and changes in regulations require them to adapt to remain relevant. Abrahamsson et al.(2002) states that for agile, one doesn't need to follow every principle and method but that projects should be tailored for their specific contexts.

In the development context of this project, customer satisfaction is paramount, and their priority lies with the customer rather than the end-user directly. As demonstrated in the results, end-users influence changes, as customer complaints, for example, are taken into account and have a significant impact. However, this intermediary layer diminishes the relevance of user involvement methods emphasized as crucial for the system's success in the preliminary project.

There is a kind of abstraction that leads to changes in methods and needs. Although this method works well, and they likely have satisfied customers, it may result in missing out on certain functionalities. As mentioned by Fernández et al.(2017) regarding requirements engineering, one of the contributing factors to various issues, such as incomplete or hidden requirements, was the customer's lack of clarity regarding their needs and desires. Developers in the project believe that investing time and resources in user-centred methods could enhance the product's quality, thus providing a competitive advantage in the market. Introducing these approaches could give more insight and therefore uncover and aid users in discovering their needs and making clear requirements.

This can also be relevant for Case C, where they go through changes with the client approving the work before implementation. The fact that the client is satisfied with the work done does not necessarily mean that the end user will be satisfied, which can create challenges. For example, suppose the client receives a lot of feedback that the work done by the consultants is not well-received by the end users. In that case, it can have negative implications for the consultants or the consulting company, even if the changes were thoroughly reviewed and the client was delighted with the outcome, as mentioned by the designer in Case C, on one of the projects they need to engage with more relevant customers, in this case, older individuals. If they solely listen to the client, a man in his 40s, they will create a system that doesn't fit the target audience for which the application is designed, resulting in an application not being aligned with the actual users.

It should also be considered whether it is possible to introduce methods such as hypothesis testing, as the context imposes limitations on how much time and resources can be allocated to the project. Is data from methods such as interviews and prototype testing sufficient? Will the qualitative data from users provide enough value for it to be worthwhile for the company?

5.4 Utilizing Data for Multiple Purposes

In this section, the general use of data in various cases will be discussed, including the types of data collected, the reasons behind data collection, and how data can be applied to purposes beyond development. While challenges were faced in finding projects specifically focused on data- or hypothesis-driven development, we explore the significance of data in these cases.

5.4.1 For features

In Case D, user data is collected to confirm or refute hypotheses generated from feedback from multiple sources. Typically, a baseline measurement is established to test these hypotheses, such as the number of new user registrations. Changes are then implemented, and the number of new user registrations over time is measured to assess the impact of the changes. This workflow is similar to the HYPEX model (Bosch, 2014, p. 160). Other metrics, such as button clicks or user drop-off rates in a purchase flow, might also be measured. However, it is essential to acknowledge that these measurements may not provide a comprehensive picture of the actual situation. Introducing quantitative methods such as hypothesis testing creates challenges with validity and reliability. For example, causal validity is important to ensure that the results can be interpreted as correlation with causation (Jacobsen, 2015, p. 351).

As mentioned earlier, there are challenges in ensuring the alignment between the data collected and its interpretation. Data collection and analysis represent a field with numerous hurdles that must be overcome for this method to become more robust. Introducing new features involves various factors that can influence the results, such as user curiosity or seasonal effects. For example, higher activity and increased usage of specific features may occur during summer due to the influx of tourists. Introducing multiple changes simultaneously can create additional variables that impact the outcomes. There are also potential problems with isolating the hypothesis, as updates generally improve several aspects and introduce multiple changes rather than one specific change.

5.4.2 For testing

A developer in Case B mentions that by collecting user data, such as tracking the user account creation process, the application's robustness can be enhanced through continuous testing, identifying potential inputs that may cause crashes. This also applies to case A, where data can be used to detect errors not captured during testing. Such errors can be challenging to identify, but data collection and analysis tools can quickly identify such issues.

5.4.3 For research

In Case C, they collect data through interviews and observations in the field rather than application usage data. This data is then used for other purposes, like research. Using data for research can provide significant value in various domains. In the context of Case C, it contributes to helping children and young people with serious issues. In other areas, such as those discussed in this study, research can yield more in-depth analyses of how users interact with a product. Moreover, establishing a solid scientific foundation can help practitioners gain insight into the benefits, as well as the risks and challenges of introducing Continuous Software Engineering (Johanssen et al., 2018).

Expanding the use of data beyond development purposes can greatly benefit data-driven and hypothesis-driven development. This can help convince decision-makers to invest time and resources in implementing this method. Utilizing data collection for multiple purposes can increase efficiency through resource reuse and fostering interdisciplinary collaboration. This is important as collaboration and communication are key to successfully developing software (Dingsøyr et al., 2010, p. 62). Overall, leveraging data for multiple purposes has the potential, in my opinion, to enhance user insight and contribute to the advancement of continuous software engineering practices.

5.5 The Evolving Landscape of Hypothesis/Data-Driven Development

This approach has significant challenges, particularly in increasing the method's rigidity and ensuring privacy compliance. Overcoming these challenges, I believe, will result in applications that are more competitive and can quickly adapt to users' changing needs. Both designers from case D expressed great interest in the method and believed it to be of significant value to them. I don't think this method will completely replace more "traditional" user-centred qualitative methods, such as interviews, observations, and user testing. Ideally, a synergy between the methods would yield the best results, as it would combine both qualitative and quantitative data to gain deeper insights into user needs. The reason is that I don't think this method will suit all contexts. There are so many different contexts and factors that greatly influence how software is developed, as seen in the cases in this study. Additionally, I don't believe everyone will have the opportunity to collect data in this manner; for example, if we were to do it in the project of case C, which involves vulnerable children and youth, there might be some resistance.

This approach, coupled with a continuous development model, will enable the creation of solutions tailored to users, can be quickly adjusted when new needs arise, and where problems are swiftly resolved. The methods also provide projects with limited resources by giving them an alternative approach to gaining user insight.

5.6 Evaluation and Limitations

The results are based on ten interviews distributed across four different cases, where the research objective was to understand better how developers work with data to understand their users. This has been partially achieved, but not exactly as planned. Several cases were not in the initially assumed context, where it was desirable to interview developers who closely interacted with their users and employed methods within agile/continuous development. The lack of access to developers working in such a manner resulted in including aspects in the results and discussion that may fall outside the initially defined research scope. As a result, the discussion may not provide the depth one would expect in chapter 1, but it explores various aspects in breadth.

Examining four cases makes it possible to create a deeper understanding and identify similarities and differences across cases. Additionally, much of what was expressed in the interviews aligns with the existing literature. This implies that even though there may not be entirely new insights discussed in this master's thesis, it can strengthen the current body of literature.

5.6.1 Limitations

One challenge that may affect the results is reflected in table 2, where it can be observed that the interviews were conducted relatively late in the semester. This affected the time available for data analysis, potentially resulting in specific perspectives that require deeper insights not being fully captured.

Additionally, I have limited experience in research, particularly in conducting thorough analyses. From the preliminary project, I became acquainted with the literature review and the use of the analysis tool NVivo, which I carried forward into this thesis. However, this is the first time I have conducted a thematic analysis. Another perspective is the aspect of interacting and working with other people. Contacting, communicating, and planning with developers posed a challenge contributing to tasks taking longer than optimal.

Having four different cases is advantageous for generalization and identifying correlations. However, a limitation is that I only had one representative from Case C, as having multiple interviewees would have provided greater insight and more credible data. Having just one representative limits the value and reduces the credibility of the results.

Most interviewees were developers and designers, providing a somewhat incomplete picture of the scenario. For instance, involving more individuals from management, such as team or tech leads, could have offered a different perspective on the area under investigation than what developers and designers can provide.

6 Conclusion and Future work

This section includes the conclusion of this project with some of the key findings, including implications for practitioners and researchers. Lastly, suggestions for future work are introduced.

6.1 Conclusion

The goal of this project was to do an in-depth exploration of how developers work with data to understand the users. It should be noted that the results were not entirely as planned, but it has given insight and value. This project has shown how development processes vary between four different project contexts. This has provided a unique insight into how developers work and their thoughts on development- processes and -methods. One aspect that emerged was how the different contexts affected the methods utilized. The results show that the context significantly affects and limits the choices concerning methods. For example, being able to work directly with end-users. Here are some of the key findings:

- There are challenges in defining metrics and analyzing data to create more trustworthy results.
- Collecting data about users can potentially provide value beyond requirements of engineering.
- Practitioners show optimism and potential benefits in collecting data about users.
- Practitioners see user involvement as valuable.

There is a general trend that developers understand the value of involving users and a desire to work with users to develop a product of good quality. This conforms well with much of the existing theory, especially regarding the benefits and challenges discovered in the pre-study. Developers are also generally aware of the value and potential benefits of collecting data about their users, even though case D is the only case that systematically collects data about their users. The case where this approach is utilized shows great optimism for the method, as the quantitative data based on actual user data has great potential for better aligning the product to the users and goes well with established qualitative methods. There are not only positive aspects with data/hypothesis-driven development; the method is relatively new and un-tested. Some challenges need to be overcome before the approach can be widespread, with regards to making the collection and analysis of data more rigorous and trustworthy and resolving security and privacy concerns.

For developers, understanding these benefits and challenges can help decide between adopting data- or hypothesis-driven methods for researchers, working with practitioners in understanding how to mitigate and resolve challenges and developing new approaches for better utilizing and improving the reliability and validity of hypothesis-driven methods.

If these hurdles are overcome, the value can be significant, and the potential for application to other purposes will be a crucial aspect that will aid in convincing decision-makers to implement and uncover and mitigate challenges. The results show that choosing the appropriate data to collect and analyzing the data poses a significant challenge. Developers refer to a lack of competence in statistical analysis, which results in uncertainty in the data and its corresponding conclusions. Lastly, although the project had some uncertainty regarding the validity and certain limitations, I still believe that the results genuinely depict the situation and that the conclusions drawn are valid.

6.2 Future work

Future work should focus on working with cases that apply some practices from data- and/or hypothesis-driven development, as this field requires more research. Several challenges are related to these methods, and more case studies focus on understanding how developers utilize this method. Another aspect could be looking specifically into how improving statistical analysis could benefit developers and increase the overall quality of the methods. Additionally, supplementing case studies with observations(and interviews) can provide even more insight into how the processes concerning creating, testing, and analyzing hypotheses are made in practice. Another field that should be further investigated concerns the definition of Continuous Software Engineering (CSE). As shown in figure 4, which illustrates the definition of CSE, the model is incomplete and more research is needed to understand the term better.

Bibliography

Abelein, U., & Paech, B. (2015). Understanding the influence of user participation and involvement on system success –a systematic mapping study. *Empirical Software Engineering*, *20*(1), 28–81. https://doi.org/10.1007/s10664-013-9278-4

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

Ahmad, M. O., Markkula, J., & Oivo, M. (2013). Kanban in software development: A systematic literature review. *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, 9–16. https://doi.org/10.1109/SEAA.2013.28

Alhazmi, A., & Huang, S. (2020). Survey on differences of requirements engineering for traditional and agile development processes. *2020 SoutheastCon*, 1–9. https://doi.org/10.1109/SoutheastCon44009.2020.9397492

Bano, M., Zowghi, D., & da Rimini, F. (2017). User satisfaction and system success: An empirical exploration of user involvement in software development. *Empirical Software Engineering*, *22*(5), 2339–2372. https://doi.org/10.1007/s10664-016-9465-1

Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70–77. https://doi.org/10.1109/2.796139

Bosch, J. (2014). *Continuous software engineering: An introduction* (1st ed.). Springer Cham. https://doi.org/10.1007/978-3-319-11283-1-1

de Villiers, C., Farooq, M. B., & Molinari, M. (2022). Qualitative research interviews using online video technology - challenges and opportunities. *MEDITARI ACCOUNTANCY RESEARCH*, *30*(6), 1764–1782. https://doi.org/10.1108/MEDAR-03-2021-1252

Dingsøyr, T., Dybå, T., & Moe, N. B. (2010). Agile software development: An introduction and overview. In T. Dingsøyr, T. Dybå & N. B. Moe (Eds.), *Agile software development: Current research and future directions* (pp. 1–13). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-12575-1_1

Dittrich, Y., Nørbjerg, J., Tell, P., & Bendix, L. (2018). Researching cooperation and communication in continuous software engineering. *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, 87–90. https://doi.org/10.1145/3195836.3195856

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). Devops. *IEEE Software*, *33*(3), 94–100. https://doi.org/10.1109/MS.2016.68

Ebert, C., & Hochstein, L. (2023). Devops in practice. *IEEE Software*, *40*(1), 29–36. https://doi.org/10.1109/MS.2022.3213285

Fernández, D. M., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetrò, A., Conte, T., Christiansson, M. .-., Greer, D., Lassenius, C., Männistö, T., Nayabi, M., Oivo, M., Penzenstadler, B., Pfahl, D., Prikladnicki, R., Ruhe, G., Schekelmann, A., Sen, S., ... Wieringa, R. (2017). Naming the pain in requirements engineering. *Empirical Software Engineering*, *22*(5), 2298–2338. https://doi.org/10.1007/s10664-016-9451-7

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

Gibbert, M., Ruigrok, W., & Wicki, B. (2008). What passes as a rigorous case study? *Strategic Management Journal*, *29*(13), 1465–1474. https://doi.org/https://doi.org/10.1002/smj.722

Hron, M., & Obwegeser, N. (2022). Why and how is scrum being adapted in practice: A systematic review. *Journal of Systems and Software*, *183*, 111110. https://doi.org/https://doi.org/10.1016/j.jss.2021.111110

Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is devops? a systematic mapping study on definitions and practices. *Proceedings of the Scientific Workshop Proceedings of XP2016*. https://doi.org/10.1145/2962695.2962707

Jacobsen, D. I. (2015). *Hvordan gjennomføre undersøkelser?: Innføring i samfunnsvitenskapelig metode[how to conduct research?: Introduction to social science research methods]* (3rd ed.). Cappelen Damm akademisk.

Johanssen, J. O., Kleebaum, A., Bruegge, B., & Paech, B. (2019). How do practitioners capture and utilize user feedback during continuous software engineering? *2019 IEEE 27th International Requirements Engineering Conference (RE)*, 153–164. https://doi.org/10.1109/RE.2019.00026

Johanssen, J. O., Kleebaum, A., Paech, B., & Bruegge, B. (2018). Practitioners' eye on continuous software engineering: An interview study. *Proceedings of the 2018 International Conference on Software and System Process*, 41–50. https://doi.org/10.1145/3202710.3203150

Kniberg, H. (2010). *Kanban and scrum - making the most of both*. Lulu.com.

Nekkøy, M. O. (2022). *User involvement in requirements engineering - an overview of the benefits and challenges* [Unpublished project thesis: TDT4501 - Computer Science, Specialization Project at NTNU, supervised by: Torgeir Dingsøyr], NTNU.

Niu, N., Brinkkemper, S., Franch, X., Partanen, J., & Savolainen, J. (2018). Requirements engineering and continuous deployment. *IEEE Software*, *35*(2), 86–90. https://doi.org/10.1109/MS.2018.1661332

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

Prenner, N., Unger-Windeler, C., & Schneider, K. (2021). Goals and challenges in hybrid software development approaches. *JOURNAL OF SOFTWARE-EVOLUTION AND PROCESS*, *33*(11). https://doi.org/10.1002/smr.2382

Schön, E.-M., Thomaschewski, J., & Escalona, M. J. (2017). Agile requirements engineering: A systematic literature review. *Computer Standards & Interfaces*, *49*, 79–91. https://doi.org/https://doi.org/10.1016/j.csi.2016.08.011

Sommerville, I. (2015). *Software engineering* (10th). Pearson.

Taraldsen, G. (2022). *Statistikk og sannsynlighet (statistics and probability)*. https://doi.org/10.13140/RG.2.2.35591.73120

van Oordt, S., & Guzman, E. (2021). On the role of user feedback in software evolution: A practitioners' perspective. *2021 IEEE 29th International Requirements Engineering Conference (RE)*, 221–232. https://doi.org/10.1109/RE51729.2021.00027

Yaman, S. G., Sauvola, T., Riungu-Kalliosaari, L., Hokkanen, L., Kuvaja, P., Oivo, M., & Männistö, T. (2016). Customer involvement in continuous deployment: A systematic literature review. In M. Daneva & O. Pastor (Eds.), *Requirements engineering: Foundation for software quality* (pp. 249–265). Springer International Publishing.

Yin, R. K. (2002). *Case study research: Design and methods, 3rd edition* (Vol. 5). SAGE Publications, Inc.

Appendix

A BPMN-model

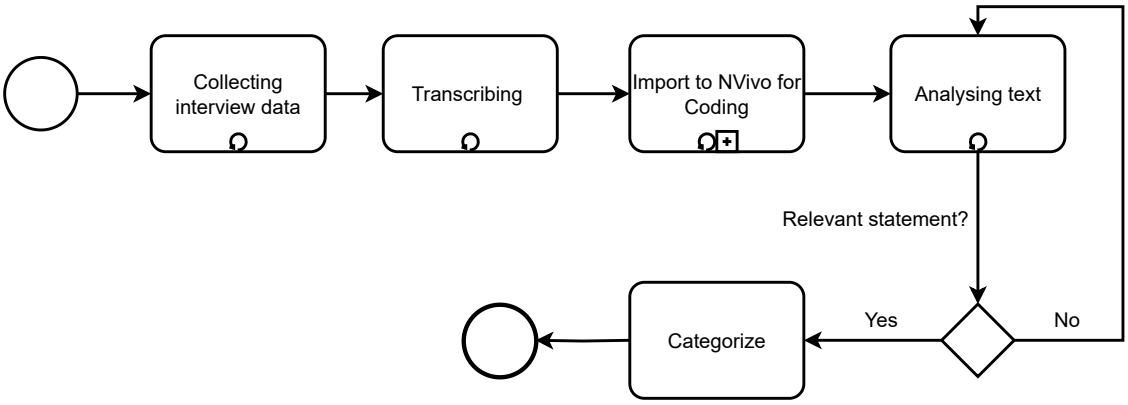


Figure 9: BPMN-model showing the different stages in collection and analysis of interview data

B Interview length

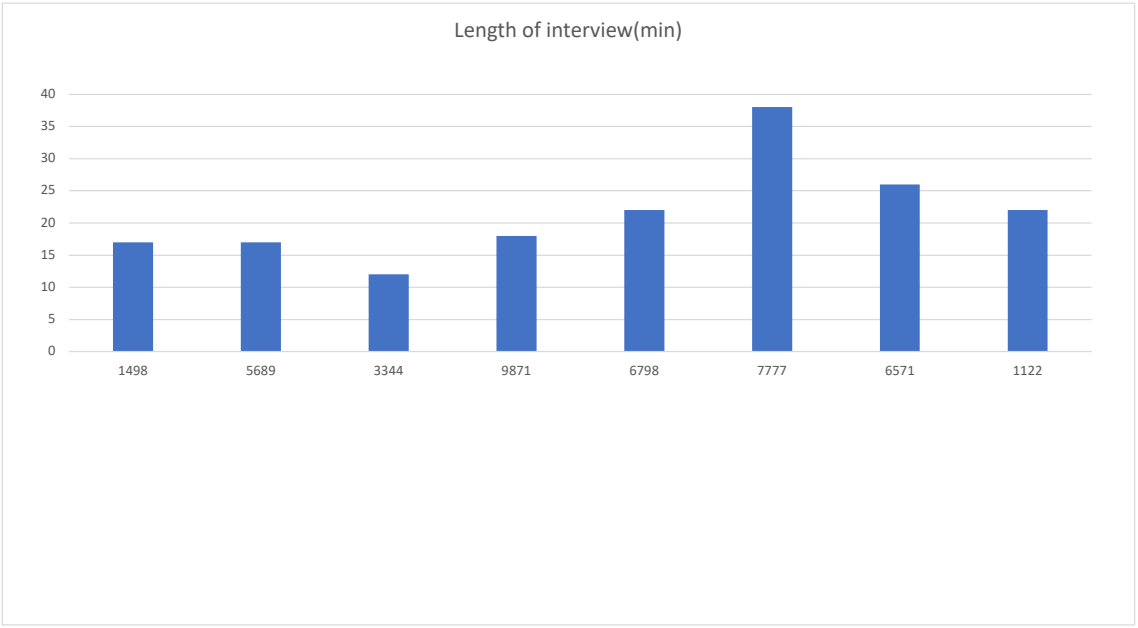


Figure 10: Length of interviews in minutes. The four numbers are the anonymised participants. The figure is sorted by date, where the oldest is the first to the left.

C Interview guide

Intervjuguide

Introduksjon

Takk for at du ville bidra til masteroppgaven!

Formålet med prosjekter å utforske hvordan utviklere jobber med å forstå brukerne sine.

Opptak av samtalen, bare jeg som har tilgang og det blir slettet når oppgaven er ferdig.
Intervjuene vil bli anonymisert slik at ingen kan gjenkjenne deg i publikasjonen.
Det er frivillig å delta, så du kan fritt trekke deg når som helst.
Det er satt av 45 minutter

Generelt:

- Bruk «Kan du utdype?» for å grave dypere.
- Nevnes data, så spør mer om det (hvilken, hvorfor)
- Generelt spør mer om hvorfor og aktiv spør etter eksempler
- Ikke vær redd for å virke dum

Del 1: Introduksjon

1.1 Fortell litt om deg selv
1.2 Hva lager dere?
1.3 Hva er rollen din? (få med hvilket team også)
1.4 Kan du fortelle litt om brukerne? (antall, roller, osv.)

Del 2: Innsamling og bruk av data

1. Hvordan jobber dere med å forstå brukeren?
    a. Eksempler på metoder eller verktøy
    b. Hva funker bra, ikke så bra?
    c. Hvorfor? (spesifikk data for å forstå brukerne)
2. Hvor ofte jobber dere med å forstå brukerne (Innsamling)
3. Brukes arbeidet til noe annet? (krav, bug-fiksing, prioritere krav, forbedre eksisterende løsninger)
4. Sammenlignet med tidligere, har det ført til endringer i hvor ofte dere snakker med kundene deres?
5. Er det ting dere har tilgang på som dere ikke bruker?
    a. Hvorfor ikke?
6. Har dere noen måte å validere avgjørelsene deres? Altså sjekke at det samsvarer med det brukerne ønsker?
7. Føler du det er noe dere kunne gjort annerledes?

Del 3: Avsluttende spørsmål:

3.1 Er det noe du ønsker å si helt avslutningsvis, noe du følte du ikke fikk sagt?
3.2 Kjenner du noen andre som kan være relevant å snakke med?

Det kan være jeg sender en mail for å få oppklaring i ting som ble sagt nå.

Plan for datainnsamling

Kriterier for intervjupersoner

Antall personer er ikke fastsatt, men det skal intervjues minst 10 personer, og maksimalt 20 personer. Intervjuene skal transkriberes og analyseres, av den grunn siktes det på rundt 10-14 personer. Intervjusubjektene må være del av team som driver med utvikling under det som kan defineres som kontinuerlig programvareutvikling, spesielt viktig er det at de samler data om brukerne sine.
Det er ønskelig å intervjue personer med erfaring, men setter ingen spesifikke krav til hvor mye.

Vurdering av metoden

Metoden blir en case study. Det blir enten fokus på ett eller flere ulike team innad i en bedrift, de detaljene er ikke fastsatt. En case study passer bra til å studere hvordan team driver med kravarbeid, grunnet at metoden er velegnet til å studere individuelle, grupper og organisasjoner på et sosialt nivå. Metoden kan blir klassifisert som en exploratory case study, siden jeg skal prøve å skape ny innsikt om kravarbeid hos utviklere som driver med kontinuerlig programvareutvikling.

Sjekkliste:

- [ ] Samler innsikt fra brukerne
- [ ] Frekvens på samling av innsikt
- [ ] Bruk av innsikten til andre formål
- [ ] Historisk utvikling med tanke på brukerne

- [ ] Bruker innsikten til formålet
- [ ] Ting som ikke blir brukt
- [ ] Validering av endringer
- [ ] Potensiale for forberedelser

D Consent form

Vil du delta i forskningsprosjektet

Data-drevet kravarbeid?

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å se på hvordan utviklere samler og anvender data i prosjekter som bruker kontinuerlig programvareutviklingsmetoder. I dette skrivet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål
Formålet med prosjekter å utforske hvordan utviklere innhenter brukerdata til kravarbeid og anvender den. Den midlertidige problemstillingen er som følger: " What are the key metrics that developers tend to utilize from the large amount of data they collect and why do they use these specific data? Prosjektet er et masterstudium som foregår våren 2023, hvor intervjuene skal foregå hos et konsulentselskap.

Hvem er ansvarlig for forskningsprosjektet?
Institutt for datateknologi og informatikk er ansvarlig for prosjektet.

Forskningsprosjektet skjer i samarbeid med Sintef sitt prosjekt Transformit.

Hvorfor får du spørsmål om å delta?
Du får spørsmål om å delta fordi din kompetanse og nåværende prosjekt er relevant for forskningsprosjektet. Utvalget skal bestå av mellom 10-20 personer fra enten samme prosjekter, eller ulike prosjekter innenfor samme selskap.

Hva innebærer det for deg å delta?
Det skal gjennomføres et intervju hvor det vil foregå et lydopptak som vil bli transkribert. Intervjuet vil ha en lengde på ca. 45 minutter. Spørsmålene vil omhandle hvordan dere jobber med å tilpasse løsningen deres for brukerne.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykket tilbake uten å oppgi noen grunn. Alle dine personopplysninger vil da bli slettet. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger
Vi vil bare bruke opplysningene om deg til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket.

- *Studenten vil ha tilgang til personopplysningene dine.*
- *Personopplysninger vil være lagret et excel ark hvor det står en randomisert kode som skal brukes for å koble lydopptak til person. Dette arket vil bli lagret i NTNU sin OneDrive hvor den vil være oppført som privat. Lydopptak vil bli overført fra personlig enhet(telefon) til NTNU sin OneDrive på lik linje med excel-arket.*

Intervjuene vil bli anonymisert slik at ingen kan gjenkjenne deg i publikasjonen.

Hva skjer med personopplysningene dine når forskningsprosjektet avsluttes?
Prosjektet vil etter planen avsluttes 5.juni. Etter prosjektslutt vil datamaterialet med dine personopplysninger slettes.

Hva gir oss rett til å behandle personopplysninger om deg?
Vi behandler opplysninger om deg basert på ditt samtykke.

På oppdrag fra *NTNU* har Sikt – Kunnskapssektorens tjenesteleverandør vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Dine rettigheter
Så lenge du kan identifiseres i datamaterialet, har du rett til:

- innsyn i hvilke opplysninger vi behandler om deg, og å få utlevert en kopi av opplysningene
- å få rettet opplysninger om deg som er feil eller misvisende
- å få slettet personopplysninger om deg
- å sende klage til Datatilsynet om behandlingen av dine personopplysninger

Hvis du har spørsmål til studien, eller ønsker å vite mer om eller benytte deg av dine rettigheter, ta kontakt med:

- *Student Mads Olsen Nekkøy (97647989) eller veileder ved NTNU Torgeir Dingsøyr (73598721)*
- Vårt personvernombud: Thomas Helgesen, thomas.helgesen@ntnu.no (93079038)

Hvis du har spørsmål knyttet til vurderingen som er gjort av personverntjenestene fra Sikt, kan du ta kontakt via:

- Epost: personverntjenester@sikt.no eller telefon: 73 98 40 40.

Med vennlig hilsen

Torgeir Dingsøyr
(Professor/veileder)

Mads Olsen Nekkøy
(Student)

Samtykkeerklæring

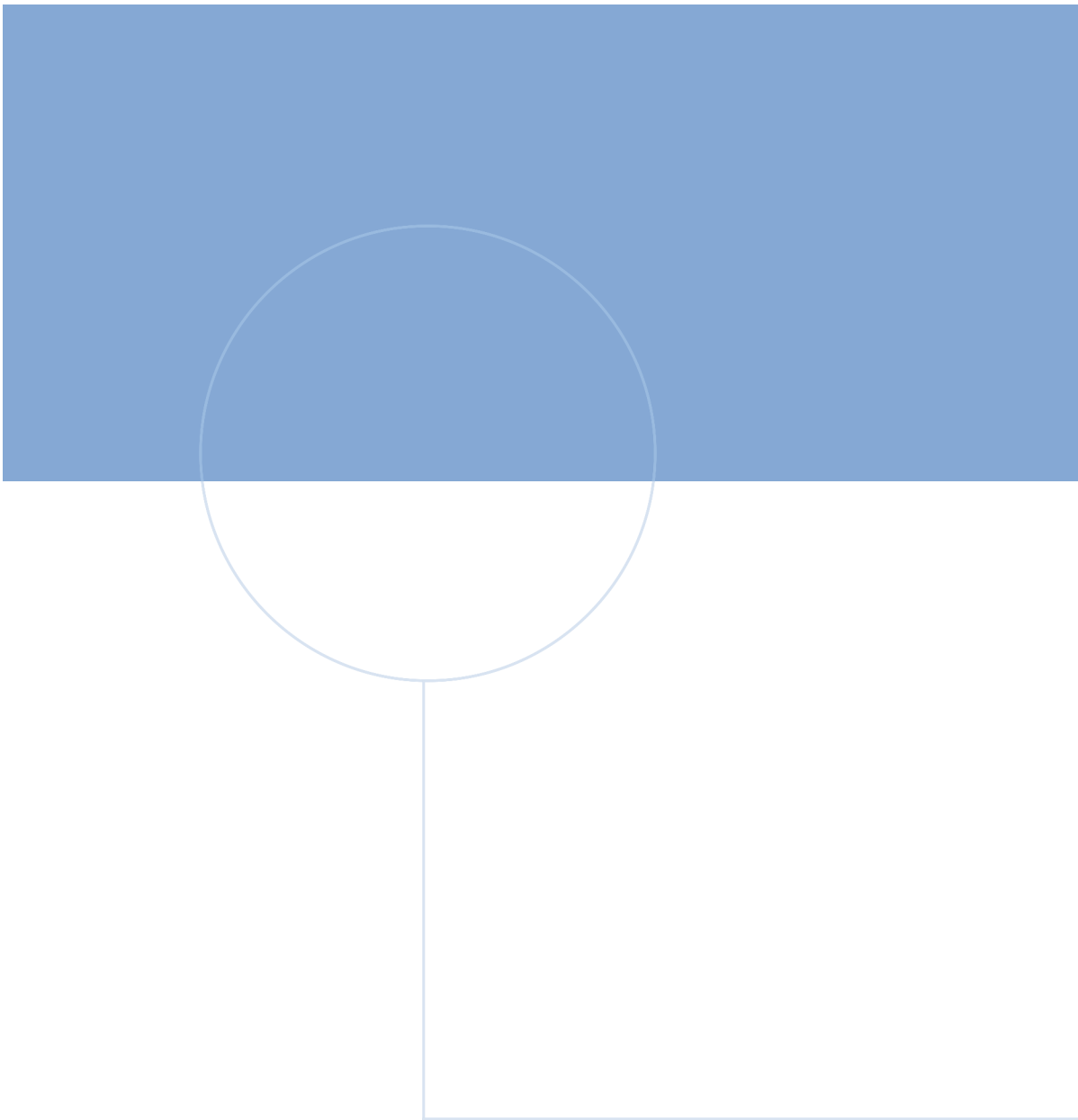
Jeg har mottatt og forstått informasjon om prosjektet *Data-drevet kravarbeid* og har fått anledning til å stille spørsmål. Jeg samtykker til:

☐ å delta i *intervju med lydopptak.*

Jeg samtykker til at mine opplysninger behandles frem til prosjektet er avsluttet

---

(Signert av prosjektdeltaker, dato



NTNU

Norwegian University of
Science and Technology