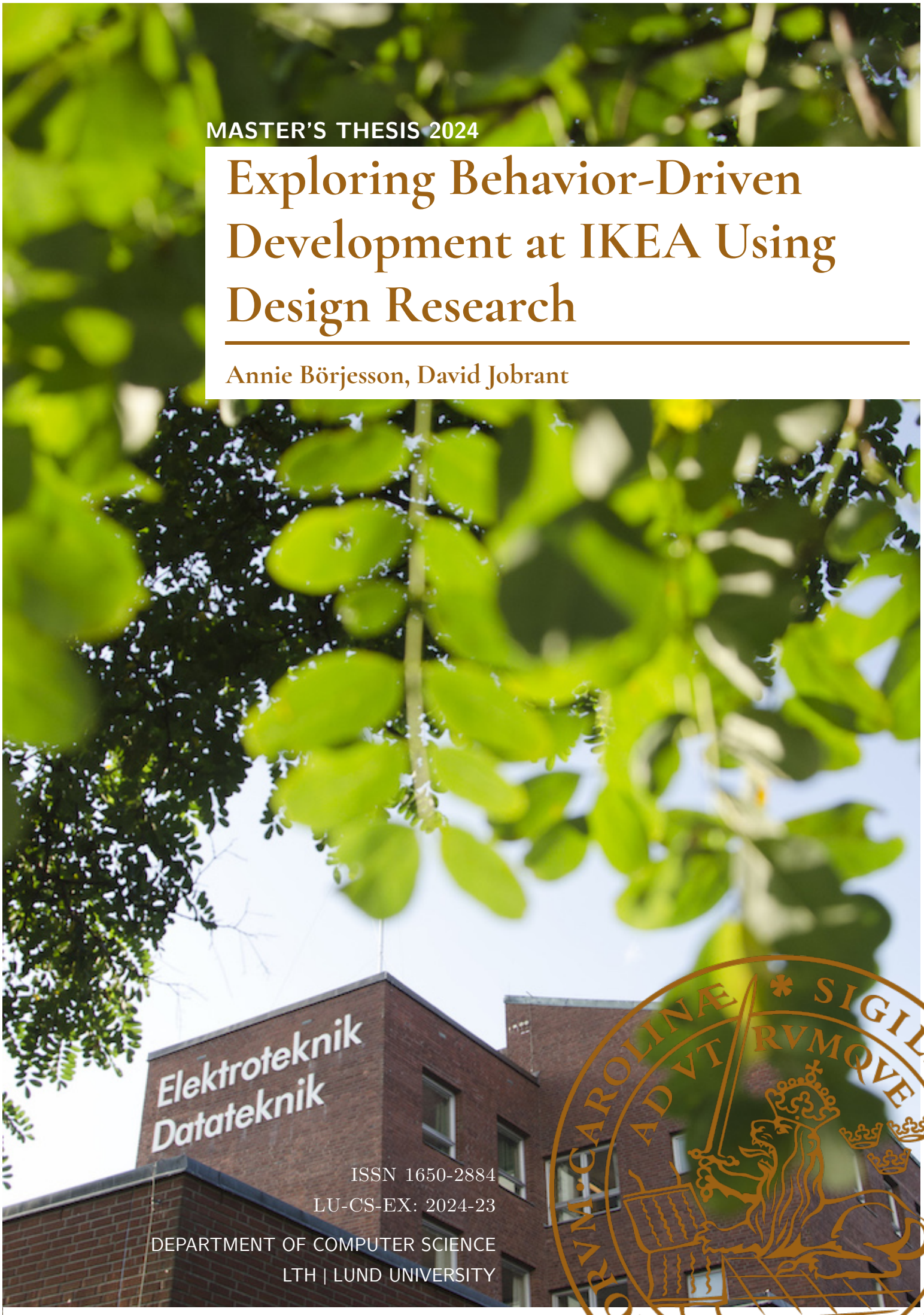
MASTER'S THESIS 2024

Exploring Behavior-Driven Development at IKEA Using Design Research

Annie Börjesson, David Jobrant



ISSN 1650-2884
LU-CS-EX: 2024-23

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-23

Exploring Behavior-Driven Development at IKEA Using Design Research

En utforskande designstudie av Behavior-Driven Development på IKEA

Annie Börjesson, David Jobrant

Exploring Behavior-Driven Development at IKEA Using Design Research

Annie Börjesson
an5416bo-s@student.lu.se

David Jobrant
da2311jo-s@student.lu.se

June 1, 2024

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Andreas Trattner, andreas.trattner1@ingka.ikea.com
Emelie Engström, emelie.engstrom@cs.lth.se

Examiner: Per Runeson, per.runeson@cs.lth.se

Abstract

In software development, testing plays a crucial role in ensuring fulfilled requirements and thereby the quality of products. However, despite its significance, testing is frequently undervalued and deprioritized due to constrained resources. This report investigates how teams within a software department at IKEA work with testing and requirements, how behavior-driven development (BDD) can be instantiated, and the impact of applying it.

We performed a study using design research consisting of three main steps: Problem Conceptualization, Solution Design and Implementation, and Evaluation, where the output of each step was utilized as input to the following activity. The problem conceptualization phase involved a multifaceted approach, including interviews as well as meetings and observations, to gain a thorough understanding of the current situation at IKEA. For the solution design, we undertook an extensive literature review to explore existing BDD instances and different aspects of BDD. Subsequently, we made an implementation following BDD principles based on the gathered insights. Our implementation was evaluated during a workshop with a team at the department.

Interview results reveal that the teams experience issues primarily related to testing and consequently to requirements. We also learned that respondents had a positive attitude towards BDD. The evaluation of our process indicates that it can be beneficial in terms of test relevance and fulfilling requirements. However, additional evaluation is suggested and described to further anchor the validity of our approach.

Keywords: Behavior Driven-Development, BDD, Software, Testing, Requirements

Acknowledgements

Thanks to Magnus at IKEA for giving us the opportunity to write our thesis in collaboration with your team. Thanks to our supervisor Andreas at IKEA; your encouragement, excitement for BDD, constant feedback and support helped and inspired us to do our best with this thesis. A big thanks to the rest of the team who welcomed us dearly and made us feel at home with all the lunches and fika. Thanks also to Linn, who organized the thesis programme this spring and took us on a trip to Älmhult to learn more about IKEA; we had so much fun!

We want to thank everyone at IKEA who has participated in our study in any way; the respondents of our interviews, the people we had meetings with and everyone else we have been in contact with.

Thanks to friends and family who have supported us when writing this thesis. A special thanks to our supervisor at LTH, Emelie. Through countless meetings, you have given us guidance, supported and encouraged us. We also want to thank our examiner Per for examining and giving us feedback.

Contents

Chapter 1
Introduction

Testing is fundamental for producing quality software. Bugs and problems are unavoidable in software development. Without testing, the issues may be hard to control and keep track of, leading to unreliable software.

Achieving a good test quality requires managing possible threats. Performing manual testing is often inefficient and leads to testers working more than they should, which in turn leads to fatigue and possibly bad tests. Another threat to the quality of the tests is communication issues. Testers who work with manual testing must be good at communication [1]. If the tester fails to understand the business requirements due to misunderstandings or lack of communication, the test quality will be reduced. Parts of the code may not be tested if the tester has not understood all scenarios which could result in a loss in the quality of the product.

Testing can be conducted in several ways using strategies with different focuses. The two main undertakings for testing are to discover issues and to demonstrate that requirements are fulfilled. Testing is thus suitable to use for quality assurance. However, getting maximal effect out of testing requires relevant tests.

Teams at IKEA have encountered challenges in maintaining the accuracy of tests and creating new ones. The issues are shared among several teams and there exists an Objectives and Key Results (OKR) group that aims to improve the test state within and among teams. The awareness of the problems with keeping tests relevant has prompted a desire to automate the testing process. Additionally, there is a desire to improve the involvement of all stakeholders at the initial and later stages of projects, anticipating that it would result in projects aligning more closely with requirements and expectations.

Behavior-driven development (BDD) has been suggested by IKEA as a possible solution to their current issues. The idea of working with it has arisen as a possible way to avoid irrelevant tests and to implement end-to-end testing among teams. Several teams share an interest in adapting BDD, which is a methodology that focuses on testing and using user scenarios to define requirements, further elaborated in Chapter 2. A central part of it is how requirements are expressed and documented. As a result, communication and collaboration

between all stakeholders are meant to be improved. BDD can be used in many different environments and industries, for example in the development of self-driving cars [2], IoT home applications [3], smart contracts on blockchain [4], game-making [5], and finance [6].

We aim to resolve current issues at IKEA and explore BDD. We will examine the current way of working with testing and requirements in a department at IKEA to determine a precise problem description. Furthermore, we will implement and present an instance of BDD in one of the teams to investigate whether IKEA's suggested solution, BDD, can be beneficial in solving the identified problems.

1.1 Research Questions

A way to determine if BDD is a viable solution is to consider test relevance and fulfillment of requirements. The scope of our research questions is the IFE-department (IKEA Family Experience), a department of software teams at IKEA. They identified test irrelevance as one of their main issues. Fulfillment of requirements can be used to ensure the quality of a product. Our thesis aims to answer the following research questions:

1. What is the current state of software testing at the IFE-department at IKEA?
   (a) How relevant are the tests?
   (b) How are requirements ensured to be fulfilled?
2. How can BDD improve the state of software testing at the IFE-department at IKEA?
   (a) Can BDD help the IFE-department to have more relevant tests?
   (b) Can BDD help the IFE-department in ensuring that requirements are fulfilled?

1.2 Contributions

Scientifically, this thesis contributes by giving insights and potential improvements to the software development processes at IKEA, as well as an evaluation of an instance of BDD in an industrial context. The results can be used in the industry as a way to improve the software development processes and testing suites at various companies.

The authors collaborated thoroughly throughout the thesis project. Both participated in the main activities such as the interviews and meetings with few exceptions. David was responsible for creating the graphics and Annie had the main responsibility for the thematic analysis. The authors were equally responsible for writing the thesis although we divided sections between the two of us.

1.3 Thesis Outline

This report begins with *Background* 2, which gives the reader details of BDD. Thereafter follows *Related Work* 3, which presents previous work related to ours. The following chapter, *Research Methodology* 4, covers how we applied the design research methodology. Then follows

three chapters, *Problem Conceptualization* 5, *Solution Design & Implementation* 6 and *Evaluation* 7, each describing one of the steps of our design research study. The subsequent chapter, *Discussion* 8, covers reflections on the answers to our research questions, as well as discussions of validity threats, generalizability, ethics and future work. Lastly, we conclude the thesis in the final chapter, *Conclusion* 9. In the *Appendix* A, our popular science summary can be found.

Chapter 2
Background

Behavior-driven development (BDD) is an agile methodology used in software development focused on describing the behavior of applications to enhance working with testing and requirements. This chapter begins with an introduction to the foundation and predecessor of BDD, test-driven development (TDD). Thereafter follows a profound description of BDD, some key aspects, and its tools.

2.1 Test-Driven Development

Test-driven development (TDD) is an agile method developed by Kent Beck as part of Extreme Programming in the late 1990's [7]. TDD emphasizes writing unit tests before implementing production code. Each test will initially fail until a developer writes code to make the test pass. When it does, the code is refactored to make sure it is clean and that duplication is avoided. By implementing a block of code after creating a test or several tests for it, various advantages over working non-interactive and writing the tests last can be achieved. One of them is that the code is more simple. When focusing on testing the functionality first rather than getting into the technical implementation immediately, it becomes easier to focus on the requirements and produce code that fulfills them. Another benefit is that when only writing code that aims to make a test pass, all parts of the code will be tested [8].

2.2 Behaviour-Driven Development

The approach used in TDD is the foundation of behavior-driven development (BDD). In BDD, just like in TDD, tests are written before implementing the actual code. However, the main focus of the methods is relatively different, as the names reveal. While TDD is all about testing, BDD emphasizes the behavior of the application. This means that discussing requirements is highly prioritized, leading to higher user acceptance than TDD.

BDD was first introduced by Daniel Terhorst-North in 2006 [9]. To tackle misunderstandings from TDD, he emphasized expressing tests as behaviors to make them more understandable and accessible. Terhorst-North states that source code should be written in a readable behavior-oriented specification, where method and class names indicate what they do. Central aspects are that acceptance criteria are expressed in tests as executable behaviors because when a test fails, the reason for it should be intuitive. The behaviors are templated using a ubiquitous language, or a domain-specific language (DSL), i.e. a language whose constructs are derived from a domain model.

Today, BDD is widely used. A study from 2020 of 50,000 GitHub projects found that BDD tools were used in 27,25% of them [10]. Results from a survey of 75 people who use BDD show that adopting BDD is usually optional and not mandatory by organizations [11]. However, 50% of the respondents said that they perceived the use of BDD in projects as very important or important. Additionally, concerning future projects, almost half said that BDD was going to be used, and more than 25% said it would be used as a key method.

BDD encourages an iterative process where business requirements are broken down into behaviors and scenarios, templated and described using plain text. Chemnitz et al. state that since BDD requirements are based on user scenarios, they represent acceptance criteria from stakeholders [12]. Hence, BDD scenarios contain quality assurance, and once the production code is implemented, it is directly accepted by stakeholders.

The usage of a ubiquitous language is a central aspect of BDD. Its purpose is to give all stakeholders, whether from a technical or business background, a shared understanding of the project [13]. The gherkin language is widely used to write BDD scenarios and is supported by tools such as Cucumber and SpecFlow [14]. An online poll performed in 2020 by Gojko Adzic, the author of the book *Specification by example*, showed that writing the scenarios on the form *Given, When, Then* (GWT) was the most popular [15]. GWT-clauses are the foundation of the gherkin language. The scenarios are written in *Feature Files*, and each file consists of several scenarios together describing specific behavior of an application.

A scenario covers one behavior and exemplifies it. GWT-clauses consist of three main parts: *Given* defines the context, *When* expresses an action or event and *Then* formulates what the expected outcome of said event is. The statements can handle scenarios at different levels of detail, and additional conditions using keywords such as *And* and *But* can be applied. A simple structure of a scenario can be seen in Table 2.1.

Scenario:	Defines the example
Given	A context
When	An action or event
Then	An expected outcome

Table 2.1: Structure of a BDD scenario.

With tool support, scenarios are mapped to executable automatic tests. Examples of such tools include *Cucumber, Concordion, JBehave, FitNesse* and *SpecFlow* [16]. These tools have separate features and support different programming languages. All do not necessarily support gherkin, but other alternative languages that can specify BDD scenarios. However, the most popular one is Cucumber [11]. In Cucumber, *Step Definitions* are generated based on the GWT-

clauses in gherkin. The step definitions are the methods that define the actual tests that are executed automatically. The type signature and gherkin tags are generated, and an example of that is seen below. The method content within the curly brackets is written by the programmer.

```
@Given("A context")
public void a_context() {...}
```

BDD is often used in combination with other methodologies [14] [17]. For example, Lopes de Souza et al. combines Scrum, ontologies and BDD to address previously encountered problems using only Scrum, such as ambiguous requirements and misinterpretations of stories [18].

Implementing BDD is time-consuming work that requires team members to be ready to implement a change in their development practices. It takes time to learn how to work with BDD and get used to the practice. How much time is required to start working with BDD is affected by various parameters. The total time consumption is bigger for BDD compared to traditional testing, where tests are written last [2]. It is however clear that it enables time to be reallocated and used for other, perhaps more valuable, activities. Students who tried working with BDD mentioned that less time was spent on debating a feature, allowing more time to create user scenarios, which enabled a better collective understanding of the features among the team members [19]. Investing time to work with BDD can also lead to a faster development process and higher customer satisfaction can be achieved [20].

Chapter 3
Related Work

In this chapter, we present previous work on behavior-driven development (BDD) and describe how it is related to our work, to position our report in relation to others and examine what has already been explored.

Several case studies on BDD have been conducted to explore the current use of BDD. Binamanungu et al. studied the use of BDD, its benefits and challenges by surveying 75 BDD practitioners [11]. Their survey outlined the current usage of BDD in the industry, as well as attitudes towards using it. Zampetti et al. studied 50,000 projects on GitHub to investigate if and to what extent BDD was used [10]. In addition, they performed a survey where developers gave their opinions on BDD. It showed that while the respondents understood the purpose of using BDD frameworks and thought that working with them led to cleaner and more consistent specifications, the required effort and the risk of it not leading to enough improvement was too high. In this thesis, we similarly look into benefits and challenges of BDD and investigate to what extent employees have heard of or used BDD. However, we focus on introducing BDD rather than exploring the current use of it.

Introduced BDD processes can be evaluated in case studies. Scandaroli et al. performed a case study and found that a key challenge when adopting BDD was to engage both the developers and the business stakeholders [21]. The authors investigated how two adoption processes of BDD unfolded in two different teams, one where the initiative came from the development side and one where it originated from the business side. They used slightly different BDD processes adjusted to their way of working. When adopting the processes, challenges identified in both teams were that engagement was low, and product owners did not design requirements clearly or in a correct BDD format, leading to ambiguity being introduced when they were translated by the developers. Some developers also found it frustrating when BDD scenarios took longer to implement than the feature itself. However, with time the workflow stabilized and everyone thought the process yielded several benefits: better communication, better alignment of requirements, confidence in tests and implementation and better documentation. On an abstract level, their process is similar to our implementation and yields the benefits we hope to accomplish.

Action research can be used similarly to case studies but with more focus on solving immediate problems and improving specific areas. Natarajan and Pichai conducted two action research studies in a scrum team, one study to define a metrics framework and one to adopt BDD practices [22]. The first study aimed to establish a comprehensive metric framework for assessing the performance of a Scrum team. In the second study, the focus shifted to adopting BDD principles alongside the team's agile methodology. The team had identified areas for improvement that aligned well with BDD principles. It included communication gaps, varying interpretations of user stories, lack of detailed discussions with concrete examples, and inadequacies in user story documentation. An instance of a BDD process was created and followed, and it improved the above-mentioned areas and also increased automation according to the team members. In-sprint automation served as a key metric to measure the impact of BDD, and it showcased a substantial increase in the number of test cases automated per sprint. In our study, we identify similar issues to Natarajan and Pichai which could be improved using BDD. Just like them, we propose a process suitable to the needs of the team at hand.

A significant amount of related work treat automation related to BDD. Alferez et al. proposed a UML-based modeling methodology together with an automated solution to generate acceptance criteria in the Gherkin format [23]. The modeling had to be done according to a certain structure in UML and was done manually, whereas the creation of Gherkin acceptance criteria was automated. A design research study was performed by Gupta et al., where the generation of conceptual models from BDD scenarios were investigated [24]. A template was used for writing BDD scenarios. Our focus regarding automation will not only be on the creation of Gherkin statements, but also on automating test results and several parts of our BDD implementation.

Automatic test generation is one aspect of automation that is the center of several papers. Raharjana et al. created a GUI tool to write scenarios and generate test cases in Codeception, a PHP testing framework [25]. They focused on entry-level programmers, hence the GUI, and accelerated the process of moving from user stories to tests. However, the tool itself was quite limited as it did not allow for changes in requirements. Instead, new scenarios had to be created if requirements change. Hijriyani et al. used Katalon Studio to generate test cases from BDD scenarios for a school information system [26]. Their process was as follows: Specify requirements, write BDD scenarios, use the Katalon recorder while performing the scenarios manually to generate tests and lastly run the tests in Katalon. The authors state that 25 BDD scenarios were successfully implemented and 23 out of the 25 automated tests passed. The ones that did not pass were caused by misuse of APIs, something that was easily fixed manually. Alferez et al. have automatically generated Gherkin specifications from a system specification in UML [23]. Using the result from a case study, the authors argued that the extra work put into modeling the system created a lesser need to rework the requirements, which was helpful due to how often requirements change in software development. Our work also includes automated generation of test cases.

Writing BDD scenarios can be done using different low-code tools, which aim to facilitate the task. An alternative to writing BDD scenarios was presented by Patkar et al. [27]. The authors wrote that stakeholders perceived writing BDD scenarios as an overhead and performed an analysis of existing BDD tools which showed that they all lacked features needed to engage non-technical stakeholders. Therefore, they suggested an approach of interacting with graphical interfaces rather than letting stakeholders write the scenarios themselves. No

evaluation was presented in the report. Another attempt to improve the scenario writing was presented by Lubke and Van Lessen [28]. BDD via BPMN (Business Process Model and Notation) has been applied as a visual tool to model requirements and tests for business analysts and developers alike. The analysts were already using it which made it easy to introduce. It led to automated tests and shifted the focus from small unit tests to larger more comprehensive integration tests, improving the currentness of tests and their maintainability. In the authors' experience, using BDD with BPMN led to better modeling of test cases and better communication between the involved stakeholders. In our study, we do not specify how the requirements are written. However, the positive results from Lubke and Van Lessen's report imply that using low-code alternatives for writing scenarios could be beneficial.

Studies have been made concerning how to increase the maintainability of an ever-growing suite of BDD scenarios in feature files. If not maintainable, they can be costly, introduce redundancy and introduce difficulties in getting an overview of actual test coverage, according to Binamungu et al. [29]. To tackle issues of maintainability, they introduced a duplication detection algorithm that receives a detection rate exceeding 70%. Irshad et al. similarly proposed a semi-automated approach using normalized compressions similarity and similarity rate to identify refactoring candidates in BDD scenarios [30]. Results indicated that their approach was up to 60 times faster than having a practitioner go through scenarios manually.

Maintainability issues are further approached in two articles which aim to improve the connection between source code and feature files containing BDD scenarios. Yang et al. investigated if it was possible to have better synchronization between feature files and source code files using Natural Language Processing [31]. They aimed to discover relations between feature files and their corresponding source code files and predict necessary updates, to ensure that the BDD scenarios accurately describe the behavior and to stay on top of test coverage. According to Diepenbeck et al., as a BDD project grows, test coverage tends to fall below 80% [32]. Therefore, they proposed an algorithm for generating BDD scenarios from source code not covered by tests, to stay on top of test coverage. In future work, they want to implement and evaluate it by applying it to different projects. While the maintainability of BDD scenarios is of importance, our focus lies in establishing BDD at IKEA. Solving maintainability issues as well as ensuring high test coverage over longer periods is something to consider once the proper BDD workflow has been established.

Architecturally, BDD can be integrated in various ways. Rahman and Gao emphasized the reusability of step definitions, decoupling of BDD scenarios and source code and ease of audibility when proposing an architecture for microservices [33]. While architecture is not the main focus of our implementation, it also covers the aspects mentioned by Rahman and Gao: step definitions can be reused, changes of requirements are considered and the BDD scenarios will be separate from the source code since they are located in different repositories.

A BDD-driven approach used to handle requirements and tests for multi-agent systems (MAS) was presented by Carrera et al. [34]. A tool and a process, BEAST, was proposed with foundations in the BDD methodology, adapted to suit the needs of multi-agent systems. In BEAST, behaviors are set by stakeholders and thereafter transformed by a MAS designer into agent stories, a set of scenarios that are broken down into test cases. The tool automatically generates JUnit test cases skeletons from BDD scenarios. According to the evaluation in the article, applying BDD in the context of MAS proved successful. Apart from reducing the number of lines of code needed, it benefited communication between stakeholders and the traceability of requirements to test cases. Just like us, Carrera et al. proposed a BDD process,

but their focus differed slightly from ours, not only in the respective contexts. We do not specify any BDD tools, but Carrera et al. present a tool package. A similarity however is that we conclude that the tracability of requirements is important.

We have noticed that there exists an extensive amount of related work. Our work does however fill a gap due to the context at IKEA and our two-sided focus on keeping tests relevant and fulfilling requirements. Our contributions are described in Section 1.2.

Chapter 4
Research Methodology

In this chapter, we provide a detailed description of our research methodology. First, we explain the choice of research method. Thereafter, we describe our implementation of design research methodology (DRM) consisting of three main steps.

4.1 Research Method Selection

We based the choice of our research method on our research questions RQ1 and RQ2. Our goal was to get a clear view of the current issues regarding testing and handling requirements and to evaluate if behavior-driven development (BDD) can improve the original state. We planned to investigate the existing problems, and based on our findings, we wanted to create an implementation of BDD that we could evaluate to some extent.

Several methods were considered when we selected a research method suitable for our goal. We considered *Action Research*, *Case Studies*, and *Design Research*. These are described by Säfsten and Gustavsson [35]. Action research is described as driven by a strive towards change through an action to identify and solve problems, and usually involves the researcher in the practitioner context. The authors further define a case study as a study of a certain matter which is suitable when aiming to gain a deep understanding of a situation or an event, or when specific aspects of them are of interest. The third method we considered, design research, includes an artifact created by the researcher aimed to be useful for the target group, according to Säfsten and Gustavsson. The contribution of the artifact is lastly evaluated.

Our plan aligned well with the characteristics of action research since it handles a practical problem in its natural context and we were focused on what IKEA wants to change and aimed to provide some sort of change. However, as we were not employed at IKEA, we were not participating in the researched context and therefore chose not to consider our study as action research. Case studies emphasize studying phenomena in a certain setting and generating theories, not necessarily designing and evaluating phenomena in that context. Design research is similar to case studies, but adds the aspect of creating and designing an artifact.

It is stated by Wohlin and Runeson to be a suitable research choice when designing concrete artifacts and extracting knowledge through them [36]. Since our plan aligned well with both Wohlin and Runeson's and Säfsten and Gustavsson's descriptions of design research, it most closely resembled our intended process. Hence, we chose to implement DRM.

4.2 Design Research

After choosing to use design research, we decided to adhere to guidelines included in design science, described by Runeson et al. as a *"research paradigm that helps frame research and aims to improve an area of practice"* [37]. The pattern of design science was applied to our design research. Runeson et al. list the research activities of design science: *Problem Conceptualization*, *Solution Design*, and *Evaluation*. These activities match the steps of design research methodology (DRM) described by Säfsten and Gustavsson [35].

Figure 4.1 describes how our design research is structured. The main activities of design science constitute the three blocks of the figure. The arrows indicate how we used the result of one step as an input to the next. The brackets on the right-hand side describe which activities concern which research question. In the Problem Conceptualization phase, we collected data through interviews and meetings and observations, which were analyzed and used as input for the following activity, Solution Design and Implementation. This phase started with a literature review, in which findings were added to the previous input used for the implementation. The implementation was then evaluated in the third phase, where we conducted a workshop and later proposed further evaluation. Each phase is further described below.

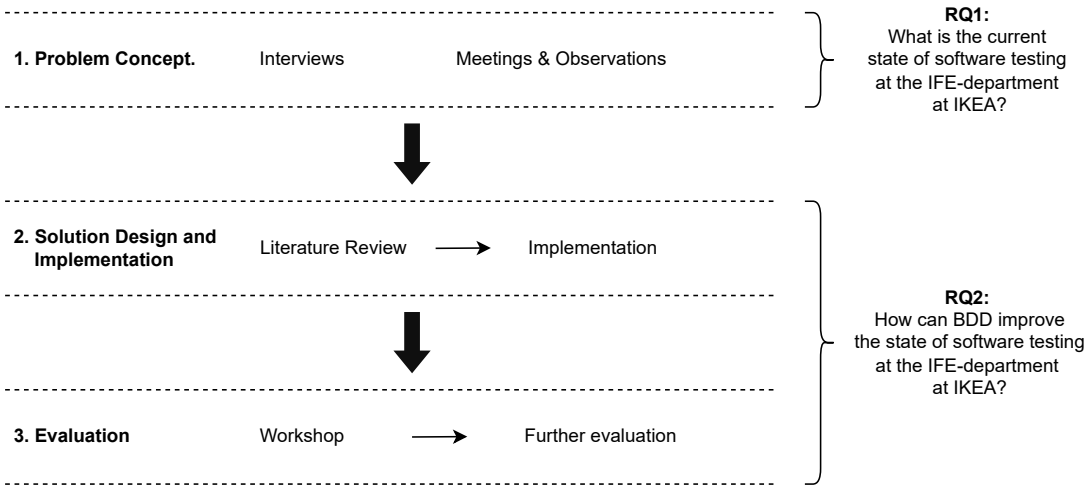


Figure 4.1: A model of our implementation of DRM.

Problem Conceptualization

The first step, Problem Conceptualization, was performed to investigate the current state at IKEA and gain a comprehensive understanding of their problems regarding testing and

requirement handling. We chose to conduct interviews since we wanted extensive first-hand empirical data from employees at IKEA. They were performed with members of the IFE-department (IKEA Family Experience), the department which is the scope of our thesis. Meetings were performed to gain broad and team-specific knowledge directly from team members of one of the teams at the IFE-department, hereby called the CRM-team (Customer Relationship Management), combined with the knowledge retrieved from the observations of their documentation and test repository. Meetings also gave us information about the Test Objectives and Key Results (OKR), an initiative at IKEA. The activities helped us answer RQ1.

Interviews
Our research questions directed the focus of the interviews toward specific topics, i.e. testing and requirement handling, and guided the formulation of interview questions. These interviews were conducted across various teams within the IFE-department, aiming to gain practical insights into the identified problem areas. The interviews are further described in Section 5.1

Meetings & Observations
To further widen our knowledge of the current state, we invited an employee from the Test OKR. We also had frequent communication with our supervisor at IKEA and a few meetings with a tester within the CRM-team who worked with BDD. Observations of the tester's code and the CRM-team's documentation were performed. This is described in Section 5.2.

Solution Design and Implementation

The second step consists of both a solution design and an implementation. The goal of the solution design was to map the problem identified in the first step to a general solution, i.e. some implementation of BDD. We therefore performed a literature review where existing instances of BDD were investigated. Leveraging the gathered insights, we created an implementation to solve the problem identified in the first step.

Literature Review
The literature review was conducted to gain insights from existing implementations of BDD. We formulated questions that stated what we wanted to answer with our literature review. The questions were based on the interview results and were considered when choosing a search term. The procedures and results of the literature review are described in Section 6.1.

Implementation
We proposed a BDD instance based on all the previous steps, focusing on automation aspects and adhering to existing ways of working and tool usage. It is described in Section 6.2.

Evaluation

The third and last step, evaluation, was to validate whether our solution was applicable in its context and solved the issues identified in our problem conceptualization. We chose to

perform a workshop with an attached survey to get an opportunity to present the implementation and effectively collect opinions on the implementation from the CRM-team, i.e. the target group. A suggestion for further evaluation is included since we find it valuable to conduct a more extensive evaluation, to which the workshop results can be used as input.

Workshop
The workshop session was performed to evaluate whether our implementation of BDD could solve the previously identified problems and answer RQ2. The workshop is described in Section 7.1.

Further Evaluation
Due to time limits, a comprehensive evaluation was not conducted in this report. A description of how further evaluation could be performed is described in Section 7.2.

Chapter 5
Problem Conceptualization

This chapter describes the first step of our design research, the Problem Conceptualization. First, we describe how the interviews were conducted and how they were analyzed. Then follows a presentation of the results. Its key takeaways are concluded and discussed in the following subsection. The second part of the chapter covers the meetings we had and the observations we made. It starts with a method description, then follows a presentation of the result, and lastly, it concludes with a section highlighting and interpreting results. The chapter concludes by listing and discussing the requirements for the solution design and implementation.

5.1 Interviews

Interviews were performed with 16 members from five different teams of the IFE-department to gain an understanding of current workways, existing issues and ambitions going forward. The results are used as input to the second step of our research design, the Solution Design and Implementation.

5.1.1 Method

The methods chosen for conducting and analyzing the interviews influence the arrangement of questions and processing of the collected data. The following section outlines the methodological approach we utilized in the interview process.

Setup of Interviews

The interviews we held were semi-structured. It is the most common way to structure interviews in engineering science according to Säfsten and Gustavsson [35]. We chose to set up the interviews accordingly to be able to change the order of the questions based on the

respondent's answers. Another advantage was that we could ask follow-up questions if a respondent talked about an interesting topic that we had not prepared questions for. On the contrary, we could also skip questions if it became obvious that they were not relevant for some respondents.

During the interviews, one researcher asked the questions, focused on the respondent's answers, made sure the conversation was kept on-topic, asked follow-up questions to cover all topics, and skipped questions if they had already been answered. The other researcher was responsible for note-taking and focused on writing down as much as possible of the answers from the respondents. Each interview was around 45 minutes and once time was running out, the researcher asking the questions was responsible for winding down the interview.

Half of the interviews were performed physically at IKEA in their Malmö offices. Due to some teams having members scattered across the world, the remaining eight interviews were performed online through Microsoft Teams. Either way, Teams was used to record and automatically transcribe the interviews. Recording the interviews is recommended by Runeson and Höst as note-taking and automatic transcriptions might not catch all details [38]. Such was the case for us, the transcriptions did not catch all sentences and occasionally made some miss-transcriptions, but due to our time limit, we decided to allow transcriptions that were not replicas of the interview. However, if something did not match between the notes and the transcription, or if the note-taker missed something, we used the recording to go back and correct our notes to make sure that we had a sound understanding of the respondent's answers.

The second to last interview was held by just one researcher, due to an impediment. The questions were asked as usual, but the notes were taken after the interview by the researcher who could not attend. The process became more time-consuming, but because of the recording, no information was lost. It was beneficial that many interviews were held already since both researchers were used to the process and could perform the interview even with a slightly different setup.

The questions of the interviews were ordered according to the time-glass principle, described by Runeson and Höst [38]. After asking some initial background questions about the respondent and their team, we asked in-depth questions related to the topic. These began with open questions, where we wanted descriptive answers. After that, we asked more specific questions, including short questions that respondents were asked to answer with a scale from 1-5. Thereafter, the character of the questions once again went towards being more broad and focused more on attitude instead of specifics.

Selection of Respondents

The demographics of the 16 respondents within the IFE-department are presented in Table 5.1. Product owners (PO), developers/engineers, and in some cases also engineering managers (EM) were interviewed. POs are responsible for the product and its delivery, focused on the business aspects. EMs are responsible for the developers and have some technical insight. Developers are the ones coding and creating functionality based on directions from POs and EMs. We had respondents from five of the 19 teams within the IFE-department. From every team, 3-4 respondents representing at least two different roles were picked. We found this amount of respondents sufficient to identify trends and draw conclusions.

ID	Team	Product nature	Role	Years at IKEA
R1	TM1	Fullstack	Product Owner	13
R2	TM1	Fullstack	Software Engineer	6
R3	TM1	Fullstack	Senior Software Engineer	7
R4	TM2	Frontend	Product Owner	4
R5	TM2	Frontend	Junior Software Developer	1.5
R6	TM2	Frontend	Software Engineer	3
R7	TM3	Backend	Product Owner	2.5
R8	TM3	Backend	Junior Software Engineer	1.5
R9	TM3	Backend	Senior Software Engineer	5
R10	TM4	Backend	Product Owner	6
R11	TM4	Backend	Engineering Manager	8
R12	TM4	Backend	Software Engineer	6
R13	TM5	Fullstack	Product Owner	2.5
R14	TM5	Fullstack	Junior Software Engineer	1.5
R15	TM5	Fullstack	Junior Software Engineer	1.5
R16	TM2, TM5	Fullstack	Engineering Manager	3.5

Table 5.1: Demographics of respondents.

Rights of Respondents

We wanted to respect the privacy of the respondents. Therefore, before starting to ask the questions, we asked if we were allowed to record the interview. They had access to both the automated transcription and the recording of the interview afterward. The respondents were informed that they could withdraw their participation at any time. We also said that they have the right to change, add, or remove content from their interviews. The respondents got information about when and where the results from the interviews would be published, and that their answers would be anonymized.

Questions

During the interviews, we obtained a thorough understanding of the current state of software testing at IKEA, which is the focus of RQ1. We asked explorative questions related to RQ2 and BDD, to determine if there already existed knowledge and interest in it within IKEA. The main questions that relate to the research questions can be grouped into a few categories:

Agile practices and communication
We asked about which agile methods were used in the team and how the communication within the team works to get a good understanding of the regular practices in the problem domain. We also asked if they used or had used TDD previously. The reason why we wanted to chart team members' experiences is because BDD builds on TDD.

Testing
The respondents were asked questions about how they work with testing, both individually and within the team. We also asked about their thoughts about the testing and the time spent on it. The reason was to collect data for RQ1 and deepen our problem understanding.

Designing and fulfilling requirements
We asked about how the respondent and its team view requirements, how they work with quality assurance and ensure fulfillment, and if they have any definition of done. As for the testing questions, we wanted to get knowledge for RQ1 and identify problems.

Behavior-driven development (BDD)
Establishing whether the respondent has used or knows of BDD already. If not, we shortly presented its key aspects to determine attitude towards using it. Experienced users were asked about their opinions. These questions are related to RQ1, but focus mainly on motivating the next step after the interviews; the implementation.

Table 5.2 lists all the interview questions in the prepared order. Apart from them, we asked some introductory background questions and for their consent in storing their data and recording the interview.

#	Question
1	Tell us about the (agile) practices within your team.
1a	Do you work according to any established agile method?
	If yes, do you do anything differently from what's decided according to that method? Why?
1b	How do you divide the work/time in? (e.g. in sprints)
1c	Have you ever worked with Test Driven Development (TDD) or Acceptance TDD?
1d	Have you considered using any other agile practices in your team?
2	How do you communicate within your team?
2a	How do you make sure that everyone has the same understanding of your product?
3	Do you use testing in your team?
3a	Tell us about how testing is used.
3b	How do you keep your test cases relevant?
3c	Why do you use testing?
4	How involved are you in your team's testing (on a scale of 1-5)? Motivate.
5	How well do you think the testing works within your team (on a scale of 1-5)? Motivate.
6	How much of your time (in one sprint) would you estimate that you/the team spend on designing requirements?
6a	How do you ensure you fulfill requirements?
6b	How do you handle changes in the requirements?
6c	How do you make sure that you don't lose quality with a change?
6d	Do you have any definition of done?
6e	Is testing involved?
7	How much of your time (in one sprint) would you estimate that you/the team spend on testing?
7a	Do you wish that more/less time was spent on testing?

7b	Which potential benefits would make you find it worth spending more time on testing and why?
8	How well would you say your understanding of your team's code is (on a scale of 1-5)? Motivate.
9	How would you compare your previous testing experiences to the testing in your team at IKEA?
10	Do you see any potential problems with your team's way of working with quality assurance?
11	Do you see any potential improvements/solutions to your team's way of working with quality assurance?
12	Have you heard anything about Behaviour Driven Development (BDD)?
12a	If yes, what would you say are the key aspects of it?
12b	If yes, what are your thoughts about it?
13	Have you worked with BDD?
13a	If yes, in your team at IKEA or elsewhere?
13b	If yes, how did you like it?
13c	If yes, how was it practiced?
14	Are you positive, negative, or neutral about BDD being introduced in your team? Motivate.
14a	Are you willing to spend time learning about BDD?
14b	What are your expectations on using BDD?

Table 5.2: Interview questions.

Analysis of Interviews

To analyze the collected data from the interviews, we conducted a thematic analysis. The goal was to find themes and categories from the data to be able to comprehend the answers. We followed the processes and tips presented by Larsson et al. in the book *Tematisk analys* [39]. It contains guidelines for performing a thematic analysis.

There is a clear connection between the research questions and the themes we found. The research questions are the basis for the interview questions. The questions about current ways to work with testing and requirement handling address RQ1 directly. RQ2 is addressed indirectly since we gather information about employees' attitudes toward BDD. The answers to the questions were coded after each interview. The codes were divided into categories, which were grouped into themes when most of the interviews had been held. The themes are therefore evolved from the research questions.

The researcher who took notes during the interview was responsible for coding that interview afterward. The interview was coded within three days after it was held, often right afterward when the interview was still fresh in mind. The codes were collected in a spreadsheet, where each column represented a respondent and each row represented a code. This layout allowed us to easily add already existing codes to new respondents when several people answered similarly. We could also efficiently see how frequent each code was since the rightmost column held the number of respondents for each code.

When coding the first interview, we worked deductively and decided on four main groups based on the interview questions; Agile processes, Testing, Requirements and BDD. The

groups were marked by color-coding rows of the spreadsheet. As we went through the interviews, we put the codes in the color blocks where they belonged. This part of the analysis was inductive. We did not limit the codes in any way, instead, we added codes that covered everything the respondents said. The analysis was therefore both inductive and deductive.

A collection of rows in the spreadsheet formed a category. These were formed when some of the interviews had been held. We divided each of the four groups into several subgroups based on the codes. The interviews that were performed after the categories had been made were coded as the previous ones, with the exception that the codes were put directly into categories. We still did not limit the codes and if no category matched a certain code, a new category was made. We did this to keep the inductive part of the analysis and maintain an open approach as we performed more and more interviews.

The codes were thoroughly reviewed once after roughly half of the interviews were performed, and once when all the interviews were completed. Codes that were not relevant to the topic of the interviews, i.e. our research questions, were removed. Similar codes were grouped to make the sheet more compact and easier to work with.

With only one interview remaining, we formed all but one of the themes. The grouping of categories was already relatively clear because of the four color-coded groups. We made minor changes and when we had a good overview of the categories, we chose to remove some of them. The sifting was performed with the research questions in mind since we wanted every category to be linked to them somehow. The remaining categories were then reviewed to formulate themes. These are, as previously mentioned, naturally linked to the research questions because of how they came about. We made sure that the connection was still clear after creating the themes by comparing the themes to the research questions.

When writing the result part of this report, we performed an unplanned review of the codes, categories and themes. We realized during this process that we missed a theme where the agile ways of working were collected. Therefore, we created a new theme to include the codes and categories covering the subject. Some minor changes were also made, e.g. a change in the name of a category and replacements of some codes.

5.1.2 Results

Eight different themes were formed during the thematic analysis: *T1 Agile Working*, *T2.1 Importance of testing*, *T2.2 Current way of working with testing*, *T2.3 Ambitions with testing*, *T3.1 Importance of requirements*, *T3.2 Current way of working with requirements*, *T3.3 Ambitions with requirements* and *T4 BDD*. Figure 5.1 groups the themes with their corresponding categories, which is presented in this chapter. As the themes and categories are presented, some results are described in terms of number of teams and some are connected to individuals, depending on how relevant the team affiliation was in relation to the code. Other results are presented both related to teams and to individuals to present the division of respondents among the teams. All of the codes that build up the categories are not be presented due to the large amount of codes. Instead, we share a selection of the codes that were common for several respondents. A few interesting and representative quotes are also included as the content of the categories is described.

The arrangement of the themes is as follows. Theme T1 covers agile working. The divisions of the themes within T2 *Testing* and T3 *Requirements* follow the same structure. T2.1 and T3.1 cover the importance of testing and requirements management and include cate-

gories that explain why it is used. T2.2 and T3.2 describe how the respondents and their teams work with testing and requirements management, which is directly connected to RQ1. T2.3 and T3.3 include how respondents would want to work with testing and requirements management. These are therefore connected to RQ2, which focuses on improvements in test relevance and requirements. T4, BDD, is also focused on RQ2 since it investigates experiences and attitudes towards BDD.

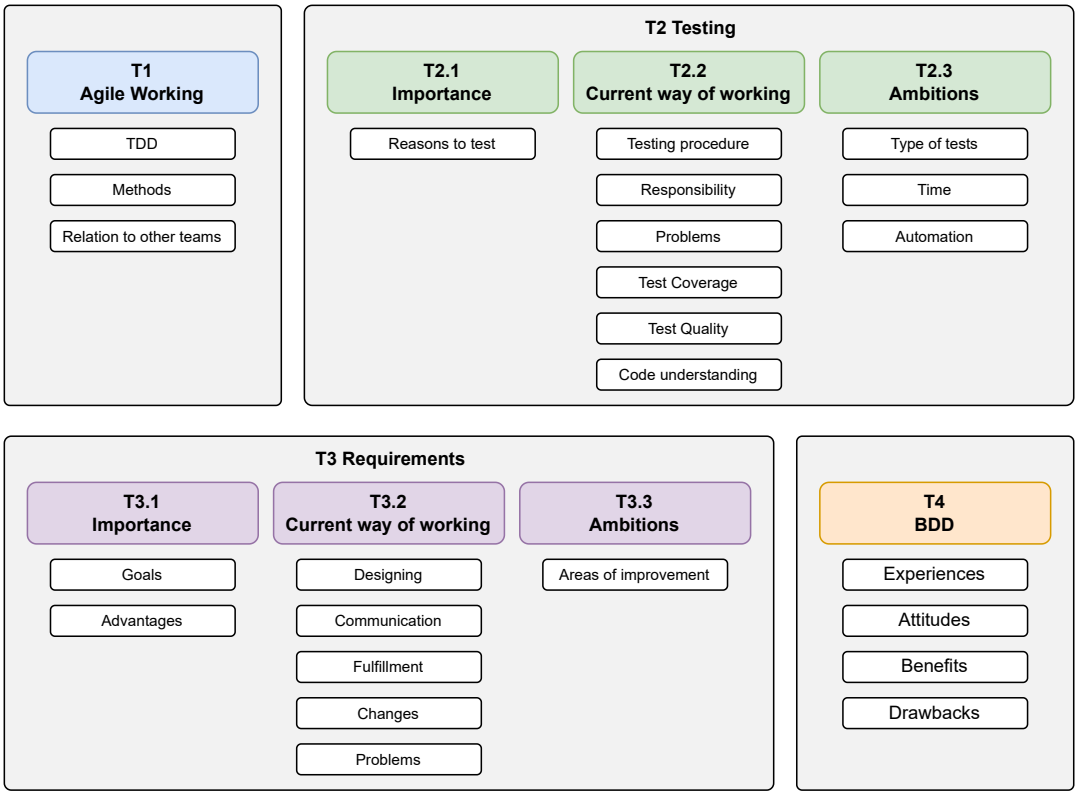


Figure 5.1: The identified themes from the thematic analysis of the interviews. Each color box represents a theme, and every white box underneath represents a category within that theme.

T1 Agile Working

We asked about the agile ways of working of the respondents and their teams. The category called TDD considers individual work, while the two other categories, Methods and Relation to other teams, focus on whole teams.

TDD
We explicitly asked whether the respondents had any experience of individually using TDD, as it is a predecessor and closely related to BDD. Eleven people said that they did not use TDD, while one respondent answered that they used TDD. R16, an engineering manager, claimed that some developers in TM2 and TM5 use TDD. Two other respondents claimed that they would want their team members to use TDD.

Methods
Several respondents from each team, in total 13 respondents, said that their team used Scrum or a similar approach. Seven of them had previously used Kanban. All of the respondents, 16 people, answered our question about sprint lengths. The most popular length was two weeks and it was used by three teams. Both one and two weeks sprints were used by one team each. Included in Scrum is the usage of stand ups, which were mentioned by 12 respondents from four teams and by three people described as a way to keep the vision of the product unified. Nine respondents from four teams mentioned that their team had a retrospective meeting at the end of each sprint to reflect on the work.

Scrum was appreciated by most respondents, but one respondent, R12, was not happy with working with sprints and explained how the way of working negatively affects the testing. Their team, TM4, had recently switched back to Scrum after using KanBan. R12 said: *"No one in our team likes sprints. [...] If it is for example only testing left of something, then testing will not be prioritized into the next sprint. Instead, the PO says 'We will take that in the next sprint after that' and we keep on pushing it forward all the time."*

Relation to other teams
Four respondents from four different teams declared that their team depended on other teams. Two of them said that it was hard and time-inefficient to test other teams' functionality.

T2.1 Importance of testing

The category below consists of codes pointing out the goals with testing and why it is performed.

Reasons to test
Seven respondents argued that testing is important. The two most frequently mentioned codes to why, with six respondents each, were *to create trust for the product* and *to detect errors instantly*. One respondent, R14, highlighted how the performance of the product is connected to the customer experience: *"It is super important that what we upload actually work as it should, since it is live towards the customers. And everyone knows that when something does not work, you get irritated and a bad customer experience."* Other common answers covered the importance of not breaking anything in production, the

want to feel secure, and the advantage that testing increases the understanding of the code.

T2.2 Current way of working with testing

The respondents were asked about how they currently work with testing and the answers we received are presented below, divided into categories.

Testing procedure
Answers regarding the current way of working that did not fit into the categories below are gathered in this category. Eight respondents said that they used unit tests. A way to work with testing used in several teams was that developers wrote unit tests for their own code after implementing it. One respondent, R6, worked in one of the teams who worked that way and said: *"Twice as much code is required for unit tests [as for UI tests] [...], while UI tests can cover extremely big flows with very little code."*

Five people from four teams mentioned that they had a blocking pipeline, meaning that tests had to pass before making a pull request and uploading the code. End-to-end testing was used and brought up by five respondents from four teams.

Responsibility
When asked about the responsibility for the testing, eight respondents from five teams said that it was up to the whole team, while three people, all from team TM1, said that the team had a designated tester.

Problems
The most common problem regarding testing was that too few tests were used due to time limits. Two respondents specifically mentioned that they had problems because of the absence of tests. One of them, R5, said: *"We have had a test set up earlier. It has above all else been unit tests. Since it took so long time to run through all these tests, we turned them off ... which has been a problem, now things break because we do not have tests that try it before we send it out to production. It is annoying not to have a test framework that we know we have to pass before going live. That is a problem."* This quotation emphasizes the problem of having too few tests and the importance of effectiveness. Another issue mentioned by two respondents was that it was hard to keep the tests updated as new features were added to the product. One PO respondent, R10, commented on the approach towards testing and said: *"Developers think 'When I am done with this component and its unit tests, I am done', but we need a bigger perspective for testing".*

Test coverage
There was no direct interview question regarding test coverage. However, six respondents mentioned it in different ways. Four of them was in two different teams and said that their team had a threshold for how high the coverage should be, which was around 80%. Two people saw issues with aiming for a specific percentage since developers often stop writing tests when the said percentage is achieved and easily forget to test all scenarios.

Test quality
The question about putting the team's test quality on a scale of one to five generated

quantitative data, is presented in Table 5.3. Rating 5 meant very good and 1 was very bad.

Rating	Number of respondents
1	0
2	6
3	1
4	4
5	1
Did not quantify	4

Table 5.3: Rating of test quality within teams.

Code understanding
Table 5.4 shows the result of the question about how well the respondents rate themselves to understand the team's code on a scale of one to five. Rating 5 meant very good and 1 was very bad.

Rating	Number of respondents
1	4
2	0
3	5
4	6
5	0
Did not quantify	1

Table 5.4: Rating of respondents' understanding of the team's code.

T2.3 Ambitions with testing

This theme consists of categories and codes regarding how the respondents would want to work with testing and their plans.

Type of tests
UI tests were something that three respondents mentioned as an ambition. Unit tests, gherkin tests, and end-to-end tests were also brought up. One respondent said that they wanted to use TDD within the team as a part of the sprint planning.

Time
Four respondents replied that they would want to spend more time on testing and four other respondents said that the time currently spent on testing was good. Three people answered that they did not want to spend more time on testing. One of them did however claim that automation could be worth spending more time on.

Automation
Seven respondents argued that they wanted to automate the testing. The automation was specified by two respondents from the same team, who described that they wanted

to use videos as user guides for their product. The videos should be updated automatically as new features are added to the application. One respondent did however say that everything does not have to be automated, and another person said that it is too hard to test everything automatically due to security protection.

T3.1 Importance of Requirements

The respondents were asked questions related to why requirements and quality assurance were used.

Goals
We had no explicit questions regarding desired achievements from the use of requirements. However, a few respondents mentioned it nonetheless. Three respondents said that the goal of having requirements was to prioritize the customer and its satisfaction. One of them elaborated and added that the focus was to make sure that changes did not impact the user negatively. Four others said that the main goal with requirements was to rely on them to make sure everything was working as expected.

Advantages
Some respondents saw general advantages to using requirements. The most commonly said things were that it gave better code quality and that developers were more self-going when they understood the needs of the end-users. One respondent, R11, put the advantage of requirements this way: *"To me, it is important that both the team and those who create new functions are aware of the importance of requirements. Exactly how they are written doesn't really matter ... but they have to exist because there has to be some form of expectation from the one making decisions, so it is possible to evaluate and understand that we're making the correct solution."*

T3.2 Current way of Working with Requirements

The following categories are related to current used practices realted to requirements.

Design of requirements
No explicit questions were asked regarding how requirements were designed, but many respondents brought it up when asked about how much time was used for designing them. The roles of the PO and developers were mentioned frequently. Six respondents from three teams said that it was the responsibility of the PO to design the business requirements, and two POs specified that it was done before the start of a sprint. One of them also stated that they discussed the requirements with the stakeholders first. Two developers in different teams claimed that their respective POs did not have much knowledge of technical aspects and the corresponding POs said the same. Four respondents from three different teams mentioned that the developers set their technical requirements, and an EM said that it was not the PO's job to care about implementation details.

Opinions regarding the clearness and existence of requirements varied. The importance of clear requirements corresponds to the size of the task according to one respondent, arguing that well-defined requirements were less important for smaller tasks.

One PO said that they had not considered requirements much, and an EM explained that they did not have requirements in place.

Communication of requirements
The requirements were commonly discussed with the team after being designed. Seven respondents from three different teams explained that requirements were communicated to and discussed with the team after the initial development. This was done through meetings, sprint plannings, refinements, and/or solution discussions. The most common way to sort out potential questions regarding requirements was to use direct communication with the responsible person, as stated by four respondents from three teams. One respondent said that the developers kept track of what to do through tasks in Jira that contained notes and descriptions from the PO.

Fulfilling requirements
Quality assurance (QA) is a way of validating that requirements are fulfilled, and the respondents explained how they worked with it quite differently. Five respondents from four teams said that they used some form of testing for QA. Two respondents from the same team mentioned that they used peer reviews for QA, and R14 said that it ensured quality: *"I think the quality is quite high, as we use peer reviews where we make sure that two developers review [the code]."* One respondent stated that their PO regularly followed up so everything worked as expected. Feedback was also mentioned by one respondent as a way to ensure that quality was not lost. Two other respondents from two teams stated that it was important to be able to communicate and receive feedback from the users easily and early.

Several answers regarding QA included tools. Three respondents from three different teams said that requirements were listed in their Jira tickets and one of them added that it worked well. Another conveyed that they had acceptance criteria for every ticket. Two respondents from the same team said that they used various methods of monitoring for QA. One of them exemplified this by saying that they used a dashboard in Power BI and the GCP console for automatic alerts as a form of QA.

In connection to the requirements questions, we asked if the respondents had a definition of done. Eight respondents, at least one from every team, said that they did, whereas five respondents explicitly said that they did not have a definition for it. That members of the same team answered differently indicates that there was not a unified view of the definition of done in some of the teams. Out of those who did not have a definition, one argued that it would be superfluous and two of them said that they avoided it because of divided opinions within the team. Teams that did have a definition of done used it differently. One respondent mentioned that there was a lot of flexibility in their definition and that it could vary between tasks. Three respondents from two teams explained that they used a combination of documentation, testing and coding as thresholds for being done. Another respondent stated that test coverage was part of their definition of done. Two respondents from different teams answered that they considered a finished ticket in Jira as being done, and another said that fulfillment of the acceptance criteria in Jira meant being done. Code review and pushing the feature to production were also mentioned in the context. Two respondents, R4 and R6 from TM2, gave short and simple answers: *"When the problem is solved it is done"*

and *"I am done when the one who made the requirement is happy"*. One EM stated that the definition was focused on what was good enough, including for example fulfilling requirements, testing and security checks, rather than what the PO expected.

Changes of requirements
Changes of requirements were handled similarly in some of the teams. One respondent said that changes of requirements could be technical or come from stakeholders. Five respondents from four teams stated that they handled changes to requirements in the following sprint. One respondent explicitly said that they were very strict about not changing the scope during the sprint. Another respondent explained in detail that if a change concerned a finished task, it was pushed to the next sprint. For ongoing tasks, the requirements were updated and changes could be made immediately.

Some teams were organized in how they handled and prioritized changes. One respondent explained that they handled changes in stories in Kanban, and prioritized them in the sprint according to urgency. Jira tickets and thorough testing were also mentioned when handling changes of requirements. One respondent noted that upon changes, code often had to get thrown away and be redone. Another respondent said that they did not currently have any proper way to handle changes of requirements.

Problems
Three respondents from two teams answered that they did not see any issues with how they worked with QA. However, a few issues were identified by others. In one team, TM4, those issues were related to their product being inherited from another team that did not have requirements set. Hence, it had been difficult to identify requirements for the product, and therefore also to connect QA to requirements. Other respondents also mentioned that requirements were time-consuming and difficult if not planned thoroughly from the start and that it could lead to a more difficult time with QA. One respondent, R12, put it this way: *"No one addresses them [the requirements] because everyone knows how time-consuming it is to establish requirements and create testing that covers all of them"*. Unclear requirements could lead to misunderstandings, as mentioned by another respondent. Similarly, one respondent argued that it was difficult to get a proper overview of requirements, especially when changes to them occurred. Another respondent identified not testing enough as one of the main issues with QA, and two others said that it would be good if they tested more. Three respondents mentioned that there was a conflict between delivery speed and quality, and prioritizing delivery speed impacted the testing and thereby the quality negatively. Lastly, one respondent said that it was not possible to measure the quality of a whole system due to time restrictions.

T3.3 Ambitions with Requirements

We identified ambitions forward when asking about eventual improvements of the requirements.

Areas of improvement
We had a question that asked whether the respondents saw any room for improvement related to requirements. While most had no answer, some respondents mentioned a

few key areas to improve. Retrospecting and long-term goals were considered necessary, according to R10, who explained: *"There is a positive attribute for me to work towards it [better QA] because then our product gets better from a long-term perspective [...] What measures I can take, is cultivating that there is a QA mindset to start with so that that mindset will help them [the developers] think about QA from the start."* Another respondent said that there existed many team-specific areas to improve regarding requirements, but did not provide any details. Two respondents from the same team said that their ambition was for their PO to write stories where the requirements were clarified as the tickets were created. This fits well with trying to make the developers understand the importance of requirements before they start implementing features, one of them added. One PO stated that they would need more formal requirements, but also did not want to overcomplicate the workflow. The respondent also added that they would need to measure more as a form of QA. Three respondents from two teams had ambitions of having traceability between requirements and testing. They had slightly different focuses, ranging from system tests to test coverage and monitoring, but altogether argued that testing would improve working with requirements. Another respondent said that their team would need good tools for creating, handling and storing requirements to improve them.

T4 BDD

The third theme is centered around BDD, including previous experiences and thoughts about implementing BDD in the team.

Experiences

Nine respondents had heard of BDD, whereof three also had worked with BDD to some extent. Two other respondents of the same team, who had not heard of BDD, described that they worked similarly to BDD but they did not label their routines. A third respondent said the same thing, with the exception that the tests were written after the implementation, instead of before as TDD and thereby also BDD suggests. One team had a designated tester who used gherkin.

Attitudes towards using BDD

A large number of codes regarding attitudes were collected. The ones with a positive approach were a majority and some examples were e.g. *A common product understanding would be helpful*, *Would like to spend time on learning BDD* and *I like gherkin* with a couple of respondents each. Four people said they were interested in BDD and one respondent mentioned that they were planning to use gherkin in the near future. Other codes were more incredulous, like *BDD can be useful, but not for all kinds of problems*, *Want to avoid too much administration* and *Do not think the productivity will increase if PO is involved in the development/testing*.

In summary, the attitudes towards starting to work with BDD were relatively scattered. What one respondent saw as a benefit, was considered a drawback for another respondent. However, the positive attitude was shared among a majority of the respondents, while the skeptical attitude was concentrated to a few individuals.

Benefits

When asked about what benefits BDD could have, five respondents replied that BDD

could involve PO more, which would be good. One of them, R15, argued: *"The PO is ultimately responsible for the product and in contact with stakeholders. If they can be connected to the technicalities too, I think it is really good."* Two respondents answered that BDD would improve communication. Other mentioned benefits were that BDD makes it easier to connect tests to requirements, avoid misunderstandings and improve structure.

Drawbacks
One drawback identified by three respondents was that it could be hard to get everyone engaged and interested in using BDD, and they saw that as a necessity to successfully work with BDD. One of these respondents, R16, was still interested in using BDD and said: *"All changes are considered badly in a team, so it requires an effort where you have to see use of the change."* Another problem mentioned by three people was the time needed for learning and start working with BDD. A disadvantage stated by one respondent was that gherkin is too logical and limits the possibility of speaking unrestrained.

5.1.3 Key Takeaways and Discussion of Results

The results from the interviews were comprehensive and provided valuable insights representative of the IFE-department. Below, we conclude the interview results and list them in order based on the current way of working, experienced issues, ambitions and BDD. We discuss coherent list points, focusing on their relationship to BDD, before we present further results.

Current Way of Working

This section presents how the five teams within the IFE-department currently work with software development. Below, we start with listing general information about their current way of working.

- Most teams used a variation of scrum with 2-week sprints. A lot of teams had previously used KanBan.
- In most teams, everyone was responsible for testing, while some teams had a dedicated tester.
- Many respondents used unit tests. Test coverage of around 80% was used by several teams as a required threshold.
- Many teams had some end-to-end tests in place.
- In a few teams, if changes in requirements occurred in a sprint, they were handled in the following sprint.
- Most POs responded that they had a low understanding of the team's code, whereas the developers overall said they had a very good understanding of it.

The above-listed results provide insight into how teams structure their work. All of these points are compatible with BDD, as it can be adapted to many different environments. Below, we continue with how requirements currently are handled in the teams according to respondents.

- Some respondents argued that requirements were relied upon to achieve customer satisfaction and ensure that the product works and behaves as intended while others had not considered requirements or did not have them in place in their team.
- Some respondents said their team had a definition of done, while others from the same team stated they did not. Those using a definition relied on requirements, testing, documentation and completed Jira tickets.
- In most teams, the PO wrote the requirements and then discussed them with the team. Developers set their technical requirements.
- Questions often arose regarding requirements and direct communication was used by most respondents to sort them out.

How requirements were looked upon varied between teams. Although many brought up its importance, some teams did not have proper requirements in place. Questions often arose regarding requirements indicating that they were not clear enough, despite discussing them within the teams. Responses on whether a definition of done existed within teams varied between members from the same teams, highlighting the ambiguity regarding quality assurance, and by extension requirements, further. This indicates that there is a need to improve requirements and organize requirements better, which is a key aspect of BDD. The following items cover how teams used testing according to respondents.

- The average estimated test quality by respondents was mediocre.
- Many respondents found testing important.
- Some teams used testing as quality assurance, while others used peer reviews or feedback.
- Most teams had a blocking pipeline.
- Most respondents did not use TDD.
- Some respondents believed enough time is spent on testing within their team, others too little, and some just enough.

Testing was mentioned to be used as quality assurance and to block merges of new code when they did not pass. Despite a common opinion that testing is important, the estimated test quality was relatively low, indicating that respondents want to improve their current testing situation and that BDD would serve a purpose. However, applying BDD could be smoother if more respondents had already used TDD, and if more respondents were open to spending more time on testing since getting started with BDD is an extension of TDD and can be time-consuming.

Experienced Issues

In this section, answers related to experienced issues of testing are concluded. We begin with listing and discussing issues regarding testing.

- Some respondents mentioned having too few tests which occasionally led to system breaks.

- Some respondents mentioned that it was difficult to keep tests updated and relevant.
- Some respondents argued that fixating on test coverage risks missing testing key scenarios.

Experiencing issues due to having too few tests was a common problem among respondents, as well as difficulties keeping tests relevant. Whether there exists a causality between the issues was not investigated, but a possible coherence is that troubles with test relevance lead to too few actively used tests, which in turn could lead to system breaks. Focusing too much on test coverage was however said to be negative. Following BDD principles is a way to achieve more tests since tests are written to cover all formulated user behaviors, without focusing too narrowly on test coverage. Further issues regarding quality assurance are presented below.

- Some respondents stated that there exists a tradeoff between delivery speed and quality, affecting quality negatively.
- Some respondents said that they did not test enough which affects quality assurance.
- Some respondents stated that they had insufficient quality assurance in various ways.

Prioritizing delivery speed can lead to insufficient testing, as insufficient time is allocated to properly implement tests. Consequently, quality assurance is negatively affected as developers can not be sure that their code works as expected. BDD assists in these issues as implementing tests are incorporated into its workflow. However, there also has to exist a will to sacrifice some delivery speed to achieve better quality. The last section of issues regards requirements and is listed below.

- Some respondents argued that frequent requirement changes prolong coding.
- Some respondents argued that requirements are sometimes unclear, leading to misunderstandings.
- Some thought it was difficult to get an overview of requirements.

Some teams lacked effective methods for handling requirements, which could lead to inefficiencies and confusion. Changes of requirements could arise from misunderstandings, or simply because stakeholders or other circumstances have changed. Either way, they can be costly and should therefore be avoided. Improved communication is a consequence of BDD, and thereby also fewer misunderstandings. For unavoidable changes, a well-defined structure is required to handle them effectively. By providing a distinct structure and by making requirements more useful in connection to tests, leveraging BDD can reduce ambiguities of requirements.

Ambitions

In this section, the ambitions of the interview respondents are gathered. We begin by listing the ambitions regarding testing below.

- Several respondents wanted UI tests that could cover bigger flows with less code.

- Some respondents wanted more end-to-end tests.
- Some respondents wanted to use gherkin and TDD for testing.
- Almost half of the respondents wanted to automate as much testing as possible.

Both UI tests and end-to-end tests are compatible with BDD since BDD focuses on users' interactions with systems. gherkin is the most commonly used ubiquitous language when applying BDD, and can thereby together with ambitions regarding TDD and test automation be achieved directly when applying BDD. In summary, BDD meets respondents' testing ambitions. Below, the ambitions of the interview respondents in terms of requirements are gathered.

- Two respondents wanted the PO to specify requirements clearly in tickets in Jira.
- Some respondents said that requirements should be more clear and easier to fulfill.
- A respondent said that requirements should be more formal without too much overhead.
- Several respondents believed that requirements should have traceability to tests.
- A respondent mentioned that they need good tooling for requirements.

While BDD does introduce some overhead, it provides a clear structure of requirements which makes them easier to fulfill by functioning as acceptance criteria. They also act as stepping stones for tests, meaning traceability between the two is provided. Several tools exist for BDD, and Jira is quite compatible with it.

BDD

Below, we conclude some opinions that were mentioned regarding BDD.

- Most respondents had already heard of BDD, but not worked with it.
- There was an overall positive attitude towards BDD. The mentioned benefits were that it involves PO, improves communication and makes it easier to connect tests to requirements.
- Most respondents wanted a low threshold for adopting new ways of working.

The positive attitude implies that respondents would like to adopt BDD in their teams. It can initially be a relatively demanding task and the threshold may not seem low enough for some respondents as a majority have not used BDD previously. However, achieving the benefits mentioned by the respondents could be argued to be worth investing in and the awareness of BDD should simplify the adoption.

5.2 Meetings and Observations

This section is focused both on the IFE-department and on the CRM-team, which is our supervisor's team. We wanted to complement the knowledge we gained from the interviews with practical insights from their daily work. Additionally, we wanted to learn more about the test initiative, Test Objectives and Key Results (OKR), at the department.

5.2.1 Method

We had weekly meetings with our supervisor at IKEA. The regular discussions were an opportunity to get valuable insights and feedback on our progress and decisions from a person with knowledge of IKEA and the CRM-team.

The CRM-team had a designated tester, whom we invited to a couple of meetings. The tester was new to the team and had started to set up acceptance tests using BDD principles and tools. Our discussions and observations of their work helped us understand BDD as well as the team's way of working better.

To gain knowledge about the Test OKR, we spoke to a person within the IFE-department active in the Test OKR. During two meetings that altogether lasted roughly three hours, we discussed and received information about the plans of the initiative, their goals and the current state of testing at the department. We also participated in a meeting with the Test OKR, to observe their plans ahead.

5.2.2 Result

In the following two sections, we describe what we learned about the CRM-team and the Test OKR from meetings and observations.

CRM-team

As also learned from interviews, the CRM-team worked with Scrum in two-week sprints. Jira was used for planning sprints and dividing tasks while providing an overview of past, current and future work. GitHub was used for version control. In production, unit tests were run according to automated workflows created using GitHub Actions, triggered when pull requests were made. At pushes to certain branches, the production code was built and deployed. When exploring the team's Jira board and comparing it to a non-enterprise Jira board, we realized that there existed limitations due to IKEA policies. Plugins such as GitHub and Cucumber were unavailable. However, scripts could be leveraged to implement similar functionality to some extent. These could be run on the internal Google Cloud Platform (GCP) server.

From meetings with the tester of the CRM-team, we learned which tools were used for testing and how it was set up. They worked in a testing repository separated from the production code, also using automated workflows in GitHub Actions. They had implemented UI tests scheduled to run regularly to determine whether a build was stable or not. This was however not activated yet for production as it was not finished. The tester wrote requirements in gherkin, using Cucumber to automatically generate step definitions. Another utilized tool was Selenium, a web driver that offers automatic UI testing. The step definitions were implemented using a Selenium driver, which automatically navigates and performs actions in the web browser when tests are run. The tester viewed the results in the IDE. The tester recently started using the test management tool Zephyr, planning to use it to structure tests and display test results in Jira in the future. Another tool that we encountered was Xray, which we discovered through observations of IKEA's documentation. It was previously used at IKEA before getting banned due to slowing down Jira.

Test OKR

The Test OKR is a collaborative test initiative at the IFE-department at IKEA. Developers from most teams of the department partake in monthly meetings. The ambition of the Test OKR is to bring people with similar expertise and interests together, share knowledge and potentially adopt some common principles. Teams within the Test OKR collaborate to determine their alignment in testing. As of now, every team optimizes separately and that can lead to trade-offs in the experience of the customer, according to the person we spoke to.

The teams have experienced difficulties testing end-to-end between teams and felt a need for automated testing to eliminate the time-consuming manual work. The person we spoke to said that 100% quality assurance is not always practical, but that they needed an element of trust so that products from different teams could be pieced together nicely. However, coordinating and engaging numerous teams has proven challenging, thus resulting in slow progress.

BDD is being introduced and recommended across the teams to facilitate end-to-end testing at the whole department. The team of the person we have spoken to uses BDD and wants to spread it in the Test OKR to improve communication and bridge the gap between all stakeholders: *"Some people in our domain are experts but not coders. Domain-driven design is about that. Given-when-then ensures a common language because it is natural and everyone understands it. We can build on the common language, and consequently build common expectations."* The idea was to start with acceptance tests in one team and its most related teams. Later on, the testing should be extended to include more teams and their dependencies. Another ambition is to move from mock data and create automated testing between teams. Those not directly involved in the initial acceptance testing had difficulties understanding their tasks, again emphasizing the difficulties in getting everyone on board.

5.2.3 Highlights and Interpretation of Results

The meetings and observations we conducted served as a valuable complement to the interview results that preceded them. Unlike the structured nature of our interviews, these interactions were more open-ended and flexible. This enabled the individuals we spoke with to express themselves more freely, offering insights and perspectives that might not have emerged in the interviews. By engaging with our supervisor, the CRM-team's tester, and the member of the Test OKR, we were able to gain additional details that enriched our understanding of testing and requirements in the CRM-team and the IFE-department. Furthermore, these interactions provided us with information of their tools usage and how we could integrate them into our implementation. Results from our meetings and observations are concluded below.

CRM-team

The following items concern what we learned about how the CRM-team currently works and which tools the team members use.

- Tools and frameworks utilized in the team were Jira, GitHub, Cucumber, Selenium, and Zephyr. The gherkin language was also used.

- The team leveraged Jira to structure sprints. The enterprise Jira had constraints that made the GitHub plugin unavailable.
- Unit tests were utilized and automatically run through GitHub Actions. UI tests based on BDD principles were also implemented.
- The team had plans for improving the testing.

The above-mentioned tools can be used with BDD. Cucumber and Gherkin directly relate to BDD, whereas Selenium can be used for UI testing and Zephyr for test management independently from BDD. Jira and GitHub are suitable tools to use together with BDD since they both include automation and integration capabilities. That the team wanted to improve testing and already had started implementing UI tests with BDD principles indicates that there is a will to establish BDD in the team.

Test OKR

These items cover the issues and ambitions that had been identified in the Test OKR.

- Experienced difficulties in collaborative testing across teams. The Test OKR wanted to set up automated end-to-end testing where BDD is encouraged to be used by all teams.
- BDD could improve communication between stakeholders of different expertise because of its domain-specific language yielding common expectations.

The Test OKR had identified issues aligning with key aspects of BDD and therefore decided to apply it to determine whether it could improve the collaboration between teams. Our interviews further indicate that several teams of the IFE-department could use better setups for testing and requirement handling, so applying BDD was a sensible decision.

5.3 Requirements on the Solution Design and Implementation

The findings from the interviews and the meetings and observations were used as input to the next step which is Solution Design and Implementation, Chapter 6. It consists of two parts: a literature review and our implementation, an instance of BDD.

We observed that automation was used in the CRM-team to some extent and it was frequently mentioned as an ambition by respondents in the interviews. Therefore, we decided to focus partly on automation in the literature review to investigate it before designing our implementation. We also decided to focus our implementation on a partly automated process rather than other important parts of BDD, such as how requirements are written. Focusing on the last mentioned parts would require more attention and time from the product owner of the team at IKEA, which was a restricted resource.

Both from the interviews and our meetings, we saw a desire to have clear and fulfilled requirements without too much overhead. BDD provides clarity in its structured scenario format, and while it might take time to get used to it, it can lead to faster development and more

satisfied customers as mentioned in Section 2.2. When performing the literature review, we decided to focus partly on requirements to emphasize their importance and find relevant articles before creating our implementation. We considered allowing flexibility when working with requirements alongside the structure BDD provides when creating our implementation.

Another aspect mentioned by respondents in the interviews was that they wanted more, relevant tests. There was also an ambition to have better traceability between requirements and tests. Providing traceability between tests and requirements ensures relevance, as long as the requirements properly describe the expected behavior of the system. Therefore, we decided to focus on traceability in our implementation.

Interview results reveal that respondents needed new processes to be easy to adapt to. Together with that insight and our observations of how the team currently works, we decided to design our implementation to align well with their practices. Tools we decided to include in our implementation were already used by the team, allowing an easy adoption. We also opted to detach our implementation from other ways of working to make it adaptable while using Scrum, which the CRM-team did.

We learned from the interviews that respondents had positive attitudes toward BDD and were interested in using it. It was also used by the tester in the team already. These findings speak for an openness towards adapting BDD. In the Test OKR, end-to-end testing was seen as an important step in ensuring quality and they have encountered problems with collaborating across teams. Using BDD can improve collaboration due to the usage of a unified language and view of testing.

To summarize, here is a concluding list of requirements for our implementation of BDD. These requirements are extracted from the interviews.

IR1: The process should result in more test automation.

IR2: The process should lead to more clear requirements.

IR3: The process should lead to better fulfillment of requirements.

IR4: The process should yield more relevant tests.

IR5: The process should include traceability between requirements and tests.

IR6: The process should be easy to adopt.

The above-listed requirements, based on the results from interviews and meetings and observation, constitute our understanding of the current situation at the IFE-department at IKEA. Aiming toward our goal of investigating whether BDD can be implemented at IKEA and the possible resulting benefits, we decided to consider the listed requirements in our literature review and map them to a solution based on BDD. This is further discussed in Section 6.3.

Chapter 6
Solution Design and Implementation

This chapter outlines the second step of our design research, Solution Design and Implementation. First, we conducted a literature review to determine how BDD has been applied previously, and what impact it had. The methodology and results are described before we conclude the review with how the results influence our implementation. The following section covers our implementation, which was inspired by the results from the literature review. To conclude the chapter, we discuss how our process fulfills the requirements of the implementation specified at the end of the last chapter.

6.1 Literature Review

We reviewed the literature on BDD, partly to investigate current implementations of BDD before designing our own, partly to retrieve information for comparison to interview results, both to strengthen our previous findings and to add other perspectives.

Some articles that we found when performing our literature review search were excluded from the literature review, but included in Related Work 3. Others were included in both Related Work and our literature review, since they were considered both to constitute interesting previous work and provide useful information in our literature review.

6.1.1 Method

Based on the results from the problem conceptualization, we wanted to discover articles about requirements, automation, relevant tools, and implementations of BDD. Kitchenham and Charters propose guidelines for systematic literature reviews in software engineering [40]. They state that the most important part of the review is to formulate questions, and that other activities of the review must be centered around the questions. They describe that the search process has to find studies relevant to the questions and that data needed for answering the questions must be extracted and synthesised so that the questions can be

answered. We followed these guidelines and formulated questions for our literature review. They are defined below with a context linking back to our previous results.

What are the main benefits and challenges of BDD?
By understanding the potential benefits and challenges of BDD, we aimed to assess its suitability as a solution to the issues identified in the problem conceptualization.

How can BDD be implemented?
By examining existing approaches to implementing BDD, we aimed to gain insights into effective implementation strategies that could be adapted to our context.

How can automation be utilized when implementing BDD?
Automation was identified as a key ambition by respondents during interviews, reflecting a desire for increased efficiency in testing processes. By investigating the role of automation within BDD implementations, we aimed to identify opportunities for streamlining testing workflows.

Which tools can be utilized when implementing BDD?
This question addresses the need for identifying tools that can support the adoption of BDD methodologies while aligning with existing processes and technologies currently used within the IFE-department and the CRM-team.

How can requirements be handled when implementing BDD?
This question addresses the challenges surrounding requirements management highlighted during the problem conceptualization. The need for clear and fulfilled requirements was a recurring theme in both the interviews and meetings.

We formulated the search string so that we could retrieve articles that could answer these questions. We extracted tools, automation and requirements as keywords for the search. The answers are concluded in Section 6.1.3.

Search String

We conducted a few searches in LUBSearch using different variations of search terms extracted from our research questions. We started very widely with just BDD as a search term, which resulted in a lot of articles that were far from our research topic. Quickly, we realized BDD is also an abbreviation for *Body dysmorphic disorder*, hence we we added behavior-driven and software to our search to ensure that only articles related to our domain were retrieved. We also added additional keywords based on the questions mentioned above. Kitchenham and Charters suggest that spelling variations should be used [40]. Therefore, we added spelling variations of behavior-driven development and the asterisk (*) was applied to cover variations of words. We placed the logical operator OR between tool*, automat* and requirement* as we were interested in articles that touched on all three subjects, not necessarily present in the same article.

The final search string used in LUBSearch is shown below. The search string was applied to all fields, including title, abstract and subject terms. Using it resulted in 124 articles.

"Behavior Driven*" **OR** "Behavior-Driven*" **OR** "Behaviour Driven*" **OR** "Behaviour-Driven*"
AND BDD

AND software
AND tool* **OR** automat* **OR** requirement*

Selection of Articles

To narrow our search and exclude articles lacking direct relevance to our literature review, we filtered the results according to the following criteria:

Inclusions
IC1: Studies that investigate benefits and challenges of BDD
IC2: Studies that propose BDD processes or workflows
IC3: Studies that utilize automation aspects of BDD
IC4: Studies that investigate BDD tool usage
IC5: Studies that investigate BDD requirement handling

Exclusions
EC1: Studies that are duplicates
EC2: Studies that are not written in English
EC3: Studies that are not peer-reviewed
EC4: Studies where BDD is not the focus
EC5: Studies outside of the software development domain

The publication year span was automatically set to 2010-2023, which we did not change as we thought it was a reasonable time frame for our search. Abstracts are often poorly written in studies in the software development domain, therefore conclusions should also be reviewed [40]. Hence, we decided to sift among the results by reading titles, abstracts and conclusions. Some articles could be disregarded by just reading the title while others were considered until we read the abstracts and conclusions. We judged their relevance to our study based on the inclusion and exclusion criteria above. This procedure resulted in 18 articles that we studied carefully and used as input to our implementation.

6.1.2 Result

We included 18 articles that were deemed relevant to our project based on the inclusion and exclusion criteria above. Nine out of these 18 are also included in Related Work 3, [11] [21] [22] [24] [25] [27] [28] [33] [34]. All 18 articles are divided into sections below, detailing contents that correspond to the title of the section. The sections are based on the questions formulated for the literature review.

Six articles described BDD processes. Four of them propose instances of BDD that could be applied or abstracted to other contexts [21] [41] [22] [34]. Automation was seldom mentioned separately from a BDD process, therefore it is included in this section. The two remaining articles identify a few considerations when implementing BDD [42] [33].

Six articles included benefits and challenges of BDD. The four articles concerning BDD processes are included, as they performed evaluations of their respective process. One of these [41] and two additional articles [20] [11] have collected opinions on the benefits and challenges of BDD in cases where BDD had already been used.

Seven articles described tools that can be used for BDD. These include the four articles presented in BDD processes, as we wanted to include the tools they leverage. The others describe BDD tools in various ways [16] [13] [26].

Seven articles described different ways to produce requirements. Three consider visual aids [27] [28] [25] and four discuss ontologies and templates as a way to structure requirements [43] [44] [45] [24].

BDD Processes

Five BDD processes, P1-P5, from four different articles [21] [41] [22] [34], are included in our literature review. We have identified their common steps, numbered from 1 to 4 and presented boldly in Table 6.1. How detailed each step was in the different processes varied, hence, we have combined smaller steps from the processes into larger more general ones in the table. We did this to more easily describe the common steps. The table also contains additional steps that were present in some of the processes but not all.

The steps of Table 6.1 are described in this paragraph. In P3, a search is performed to find existing behaviors to allow for reuse of them. In step 1, all processes define new behaviors. Typically, the product owner (PO) collects requirements from the customer and organizes them before communicating them to the team. Precisely how requirements were transformed or mapped to user stories and behaviors varied between the five processes. Additional refinement sessions with developers and testers occur in P1 and P4. Step 2 in Table 6.1 involves creating feature files with scenarios, which ranges from initially writing user stories to directly formatting BDD requirements. This task is accomplished by requirement and verification engineers in P1 and by developers in the other processes. In step 3, the testers or developers focus on test automation, defining step definitions and other code that may be necessary for testing purposes. Subsequently, source code is implemented by the developers in P1, P2 and P4. Finally, in step 4, tests are executed and results verified. The most frequently mentioned benefits and challenges of the processes described in the articles are summarized in Table 6.2. In the following paragraphs, we describe the studies of the processes.

Step	Explanation	Actor	Process
	Search for existing reusable behaviors	Requirement Engineers, Domain Experts	P3
1	**Specify new behaviors**	Product Owner, Customer	All
	Refine behaviors	Product Owner, Developers, Tester	P1, P4
2	**Create feature files with scenarios**	Requirement & Verification Engineers / Developers	All
3	**Test automation**	Tester / Developers	All
	Source code implementation	Developers	P1, P2, P4
4	**Execute tests and validate results**	Tester, Developers	All

Table 6.1: Steps of BDD Processes. The steps common for all processes are numbered and marked in bold.

Scandaroli et al. conducted a case study investigating how two adoption processes of BDD, P1 and P2, unfold in two different teams, one where the initiative came from the development side and one where it originated from the business side [21]. P1 and P2 are slightly different and adjusted to their corresponding team's way of working.

Using a technology transfer model, Irshad et al. propose a BDD process, P3, for large-scale software systems [41]. BDD was presented in workshops, and participants were asked to identify potential challenges and benefits. Thereafter, a process was suggested and evaluated

through interviews in an industrial context. The process is defined abstractly and focuses on the input and output of certain steps, and who is responsible for each step. Hence, the authors suggest a way of working that can be adapted freely from a team's internal way of working and toolset.

Natarajan and Pichai conducted an action research study in a Scrum team where BDD practices are proposed and adopted [22]. The team had identified areas for improvement that aligned well with BDD principles. It included communication gaps, varying interpretations of user stories, lack of detailed discussions with concrete examples, and inadequacies in user story documentation. An instance of a BDD process was created and followed, and the team members gave feedback on the process. In-sprint automation served as a key metric to measure the impact of BDD, and it showcased a substantial increase in the number of test cases automated per sprint. To further validate these findings, a similar but improved approach, P4 in Table 6.1, was applied to a second team. This process also resulted in a significant improvement in in-sprint automation rates.

Carrera et al. used a BDD-driven approach to handle requirements and tests for multi-agent systems [34]. A tool and a process, BEAST, was proposed with foundations in the BDD methodology, adapted to suit the needs of multi-agent systems. The tool automatically generates JUnit test cases skeletons from BDD scenarios. According to the evaluation in the article, applying BDD in the context of MAS proved successful. The process, P5, is specific to the context of MAS, but the BDD aspects are similar to the other processes. The steps that are not strictly related to BDD are excluded from Table 6.1.

Contan et al. did not propose a BDD process but instead described three needs when applying BDD [42]. These needs were: *Reusing the implementation steps for the BDD scenarios*, *Uncoupling the code sequences in business, testing and implementation models* and *Reader and user-friendliness*. Rahman and Gao also identified three needs, focused on microservices development using BDD [33]. They were: *Reusability of step implementations for BDD scenarios*, *Separation of concern among developers, testers and business analysts* and *Ease of auditability*. The steps from the two different articles are very similar, but the last steps are the ones that differ the most. While Contan et al. focused on the easy execution of scenarios, Rahman and Gao highlighted the need for easy revision.

Benefits and Challenges of BDD

Three articles [41] [20] [11] were included in our literature review, they collected and discussed benefits and challenges of BDD. We also included the articles from BDD processes that evaluate their respective processes. The most frequently mentioned opinions in the articles and aspects considered to be of relevance based on results from our interviews are listed in Table 6.2 and 6.3. Irshad et al. performed workshops investigating BDD in a large-scale context [41]. A systematic literature review is compiled by Farooq et al. [20]. Binamungu et al. surveyed 75 BDD practitioners to gather opinions about BDD, focusing on challenges regarding maintenance [11].

Tools

Several tools can be used when adapting BDD and the choice usually depends on which programming language the team uses. Table 6.4 lists the tools present in seven articles included

Benefits of BDD	Reference and nature of study
Improves understanding of requirements	Workshop [41], Survey [11]
Improves clarity of requirements	Process Evaluation [22]
Improves quality of requirements	Process Evaluation [41], Literature Review [20]
Improves requirement verification	Literature Review [20]
Improves alignment of requirements	Process Evaluation [41] [21]
Improves traceability between requirements and tests	Process Evaluation [34]
BDD scenarios clarify system behavior	Workshop [41]
Focuses on user needs	Literature Review [20], Process Evaluation [41]
Improves communication	Literature Review [20], Survey [11], Process Evaluation [41] [21] [22] [34]
Improves collaboration	Literature Review [20], Process Evaluation [22]
Increases and streamlines test automation	Process Evaluation [22]
Great reusability of code, scenarios and/or features	Workshop [41]
Scenarios act as executable requirements	Survey [11]
Improves organization of tests	Workshop [41]
Increases confidence in tests and/or implementation	Process Evaluation [21]
Improves code quality	Literature Review [20], Survey [11]
Improves documentation	Process Evaluation [21] [22]
Reduces number of code lines	Process Evaluation [34]

Table 6.2: Identified Benefits of BDD. Workshop, Survey and Literature Review cover opinions expressed in cases where BDD was already used. Process Evaluation gathers opinions on when a specific BDD process was introduced and evaluated.

in our literature review. Three specifically discussed tools [16] [13] [26]. Cucumber, JBehave, SpecFlow and RSpec are all BDD frameworks. FitNesse and Concordion are automated testing frameworks with limited BDD support. Katalon is a quality management platform that supports a wide range of testing frameworks, for example, Cucumber for BDD testing. Selenium enables browser automation which is primarily used for UI-testing. Jenkins is an automation server that automates parts of software development, namely building, testing and deployment. Table 6.5 displays which tools were used for the various BDD processes described previously.

Requirements

BDD requirements can be produced in different ways. This was discussed in seven articles included in our literature review [27] [28] [25] [43] [44] [45] [24]. Requirements can be written with or without support, or by using a low-code alternative. Patkar et al. state that stakeholders perceive writing BDD scenarios as an overhead and perform an analysis of existing BDD

Challenges of BDD	Reference and nature of study
Maintainance of requirements	Workshop [41]
Poorly written requirements	Literature Review [20]
Difficulty writing requirements	Workshop [41]
Unclear requirements from POs	Process Evaluation [21]
Might be expensive to implement	Workshop [41], Literature Review [20]
Difficulty writing test cases	Workshop [41]
Duplicated test cases	Literature Review [20]
Absence of instruction	Literature Review [20]
Changes team's traditional approach to software development	Survey [11]
Hard to quantify benefits	Survey [11]
Steep learning curve	Survey [11], Workshop [41], Literature Review [20]
It requires learning of new technologies	Process Evaluation [41]
May lower team productivity	Survey [11]
Dependence on far too many stakeholders	Process Evaluation [41] [21]
Behaviors may be hard to define for large-scale systems	Process Evaluation [41]
Time consuming to write BDD scenarios	Process Evaluation [21]

Table 6.3: Identified challenges of BDD. Workshop, Survey and Literature Review cover opinions expressed in cases where BDD was already used. Process Evaluation gathers opinions on when a specific BDD process was introduced and evaluated.

Tool	Features	References
Cucumber	BDD Framework that supports several programming languages. Uses Gherkin syntax in .feature files	[16]
JBehave	BDD Framework for Java. Supports JBehave and Gherkin syntax	[16]
SpecFlow	BDD Framework for .NET, uses Gherkin Syntax	[16]
RSpec	BDD Framework for Ruby, uses its own syntax.	[13]
FitNesse	Automated testing framework, limited BDD support	[16]
Concordoin	Automated testing framework, limited BDD support	[16]
Katalon	Testing and quality management platform. Supports BDD through Cucumber.	[26]
JUnit	Testing framework for Java.	[34]
Selenium	Web driver for browser automation, primarily used for automated testing.	[22]
Jenkins	Automation server for building, testing and deploying. Commonly used for CI/CD.	[22]

Table 6.4: Tools for BDD and testing.

tools which shows that they all lack features needed to engage non-technical stakeholders

Process	Tools used	References
P1	Cucumber (for Java)	[21]
P2	JBehave, JUnit	[21]
P3	Abstract process, no tool suggestions	[41]
P4	Cucumber, Jenkins on Docker, JUnit, Selenium	[22]
P5	JBehave, JUnit, MAS-specific tools	[34]

Table 6.5: Tools used in processes described in Section 6.1.2.

[27]. Therefore, they suggest an approach of interacting with graphical interfaces rather than letting stakeholders write scenarios themselves. Lubke and Van Lessen have also introduced a visual tool to model requirements and tests [28]. In the author's experience, the usage of the tool led to better modeling of test cases and better communication between the involved stakeholders. Another GUI tool for writing scenarios and generating test cases is presented by Raharjana et al. [25].

Silva et al. state that it is difficult to write initial requirements using existing tools, and address the issue with an ontology [43] [44]. A succeeding evaluative case study with four participants conducted by the same authors reveals that Product Owners (POs) exhibit strong adherence to the ontology, even in instances where they are unaware of its existence [45]. When presented with an example of a user story in BDD format utilizing templates from the ontology, and subsequently writing their own stories, the majority of statements (62.26%) drafted by the POs closely matched the templates.

Gupta et al. came to different conclusions when describing a template for writing BDD scenarios [24]. It was expected to result in standardized statements and the authors assumed that relevant fields would always be filled. However, a concluding limitation mentioned is that it may not be the case in reality. How to ensure scenarios are always written correctly is not specified by our implementation.

6.1.3 Influence of Results on Implementation

Here, the results from the 18 studied articles of the literature review are concluded. We also discuss how the literature review results influenced our implementation.

We have identified four key steps common in five processes in four different articles, presented in Table 6.1. These steps were: (1) Specify new behaviors, (2) Create feature files with scenarios, (3) Test automation and (4) Execute tests and validate results. Apart from these four steps, we identified two additional steps present in at least two processes: Refine behaviors and Source code implementation.

The benefits achieved from the five processes are included in Table 6.2. Benefits raised by the evaluation of these processes included improved communication and collaboration. Using BDD improved the clarity, quality, and alignment of requirements. BDD focused on scenarios and clarified system behaviors that acted as executable requirements. It improved traceability between requirements and tests, as well as automation of tests. Using the processes increased documentation and confidence in source code. Most of the benefits aligned with the ambitions of the interview respondents and are therefore benefits we want to focus on achieving. Hence, we chose to include the four common steps and the additional two mentioned above in our process.

Challenges mentioned from evaluations of the processes are that requirements were ambiguous and that BDD scenarios were time-consuming to write. Other mentioned challenges were dependence on too many stakeholders and that BDD requires learning new technologies. Including more stakeholders was mentioned as a benefit of BDD in our interviews, indicating that there are different opinions on the subject. The challenge of having to learn new technologies to use BDD coincides with our interview respondents' need for a low threshold for new methodologies. This need could constitute an obstacle when adopting BDD. However, the benefits align very well and some learning is necessary to achieve the benefits. To avoid the obstacle of learning new technologies, we decided to make our process easy to adopt by tailoring it to the CRM-team's conditions.

Maintainability is mentioned as a general challenge of BDD. We decided not to to focus on the maintainability of our process since eventual maintainability issues tend to arise when BDD already is in use and has been for a while. However, we decided to make our process compatible with maintainability measures, which can be integrated at later stages. One measure is to specify a standardized way to write requirements as that can avoid maintainability issues.

There exist several articles suggesting graphical interfaces to assist stakeholders when writing requirements in BDD format. Ontologies are also possible to use to establish a common language, ensuring that requirements are not only written in the same format but also using the same wording and structure of sentences. This could tackle the maintainability issues mentioned previously. In our process, we decided not to focus on how requirements are formulated apart from the structure BDD provides, due to time constraints. However, either a graphical interface or ontology could be used to provide further assistance and make requirements more standardized.

Table 6.4 lists tools that can be used with BDD for testing. These were Cucumber, JBehave, SpecFlow, RSpec, FitNesse, Concordion, Katalon, JUnit, Selenium, and Jenkins. We decided to search for tools in the literature review to determine what is currently feasible by BDD tools since it could be beneficial when creating our process. However, we decided to describe our process abstractly and not specify any tools in our implementation of BDD to make it more generalizable.

6.2 Implementation

Our implementation was created with input from the Problem Conceptualization in Chapter 5 and the literature review in Section 6.1. It consists of a process of twelve steps describing a workflow that follows the BDD principles. It contains all four steps present in the BDD processes studied in our literature review, gathered in Table 6.1. Our BDD process listed in Table 6.6 focuses on clarity, reusability and fulfillment of requirements, uncoupling tests from production code, traceability, test automation and a low adaptation threshold. We do not specify any tools apart from Jira and GitHub. To define a relevant process we suggested steps that were backed by our interviews. The literature review anchors the strength of our suggestions. Hence, our process is evidence-based.

We experimented with Jira to familiarize ourselves with the environment and explore existing features that could be utilized in our process. We investigated what kind of tickets existed, the relations between them, how new ones could be created, and automation. This

was done both on the CRM-team's Jira board, which was an enterprise version, and on a separate board created on the non-enterprise version of Jira. The connection between GitHub from Jira was explored, as well as GitHub Actions.

During the design of the process, we were in frequent contact with our industry supervisor. They gave us feedback from IKEA's perspective and described what would suit their work. We based the information about the needs and wishes of coworkers at the IFE-department on the Problem Conceptualization, Chapter 5, but additional aspects from our supervisor helped us tailor the process to the CRM-team.

Some steps of the process are meant to be performed by a specific role. The roles considered were product owner (PO), developer and tester. Step 1 is carried out by the PO which involves other roles as necessary. Steps 2-5 can be performed by any role, preferably by or in collaboration with the PO to involve them in the process. Steps 6 and 7 are conducted by the tester and steps 8, 9 and 10 by the developer. Steps 11 and 12 are relevant for everyone in the team. When describing and discussing the process, the following terms will be used:

- **REQ:** An issue of the type *Requirement*. REQ always contains the latest, most updated feature file.
- **Ticket:** An issue of the type *Story* connected to a REQ. Receives feature file of REQ when created, does not get updated as REQ changes.
- **Feature field:** A field in an issue of type *Requirement* and *Story*, where BDD-requirements are viewed and edited.
- **Feature file:** The content of the feature field constitutes the feature file.

Everything related to tickets refers to Jira. The issue type *Requirement* was not predefined in Jira, so we created a new type that contained a feature field. Branches are located in GitHub. Figure 6.1 describes the information flows and illustrates where various parts of the process are located.

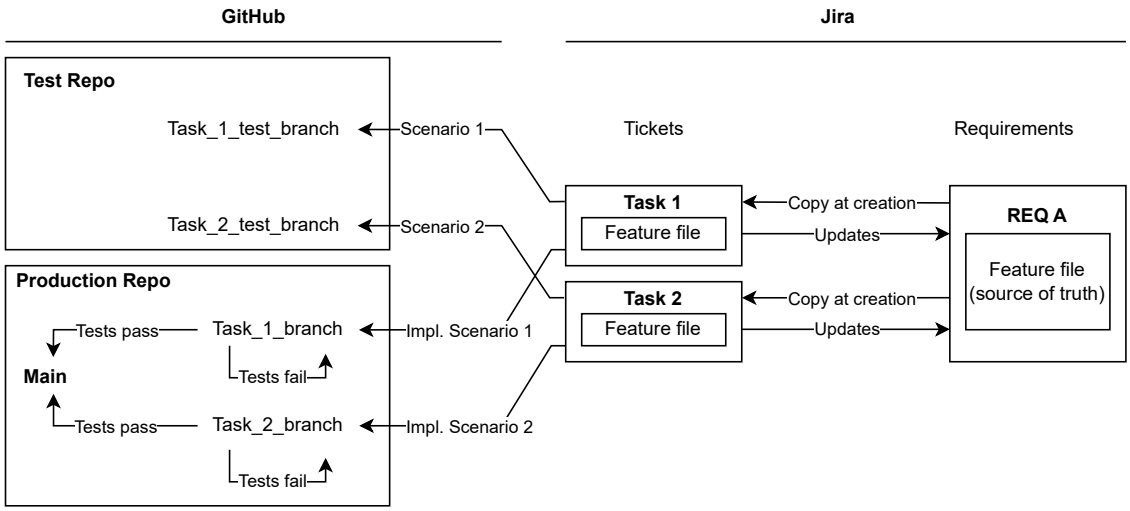


Figure 6.1: Overview of flows in our process.

Table 6.6 below lists the steps of our process, together with motivations emphasizing the reason for including each step. Motivations regarding the literature review are marked as

(LR). By considering aspects from other processes, our process could achieve similar benefits as them. Motivations regarding the interviews are focused on respondents' needs and are marked as **(INT)**.

Step	Description	Motivation	Role
1	**Specify behaviors** The requirements are discussed and set.	Corresponds to Step 1 in Table 6.1.	PO
2	**Search for existing feature file** The search is filtered on the label *BDD-feature* and issue type *Requirement*. **a) If file does exist** The existing REQ is found after searching for relevant terms and words included in the feature field of REQ. **b) If file does not exist** No REQ is found and a new one with an empty feature file is created as an issue of the type *Requirement* and labeled *BDD-feature*.	Part of process P3 in Section 6.1.2 (LR [41]). Reusability of requirements avoids duplicated REQs and scenarios (LR [41]).	PO
3	**Create a ticket and connect to REQ** The REQ found or created is linked to the new ticket. This triggers a Jira automation which invokes the next step.	Traceability between new requirement and feature file (INT). REQ is the source of truth, providing clarity of requirements (INT) (LR [41] [22]).	PO
4	**Import copy of feature file** The content of the feature field of the REQ is automatically copied into the feature field of the new ticket.	Traceability between different versions of the feature file (INT), tackling the challenge of version control of requirements in BDD (LR [41]). Automation (INT).	Auto
5	**Edit feature field** A change is made to the feature field of the new ticket according to the new requirement. The change is highlighted. As the ticket is saved, the feature field of the related REQ is updated.	Corresponds to Step 2 in Table 6.1. REQ is the source of truth, providing clarity of requirements (INT) (LR [41] [22]). Traceability between different versions of the feature file (INT), tackling the challenge of version control of requirements in BDD (LR [41]).	Tester

Step	Description	Motivation	Role
6	**Create a branch on the test repo on GitHub** The tester opens the ticket and clicks *create branch*. The feature file on GitHub is updated according to the change in the ticket.	Traceability between tests and requirements due to automatic updates of the feature file (INT) (LR [34]), which ensures correctness between Jira and GitHub. Separate repositories to uncouple tests and production code (LR [42]).	Tester
7	**Implement step definitions and merge test branch** Type signatures and Gherkin tags describing the corresponding statements (Given, When or Then) for step definitions are suggested based on the new requirement. The tester implements code for the step definitions and merges the branch.	Test automation (INT) (LR [11] [22]). Corresponds to Step 3 in Table 6.1.	Tester
8	**Create a branch on production repo on GitHub** The assigned developer opens the ticket and clicks *create branch*. A branch for the ticket and its scenario is created on the production repo on GitHub.	Separate repositories to uncouple tests and production code (LR [42]). Traceability between branch and ticket due to automatic branch naming and visibility of the branch in the ticket (INT).	Developer
9	**Implement production code** The developer implements production code on the created branch.	Code is implemented after tests to follow TDD principle included in BDD.	Developer
10	**Merge the production branch on GitHub** GitHub Actions triggers the execution of the tests at the merge and blocks it until all tests pass. A merge request is then raised and peer-reviewed by two other developers in the team. When accepted, the branch is merged and closed.	Reliance on tests facilitates the process and improves confidence and reduces error proneness along with peer reviews (INT) (LR [21]).	Developer

Step	Description	Motivation	Role
11	**View test results** Test results are visible in Jira. The results for individual scenarios are presented in their respective tickets. REQs show results for the corresponding feature file.	Visibility and validation of test results, enabling tests as quality assurance (INT). Corresponds to Step 4 in Table 6.1.	Any
12	**View feature history** The history of a feature file is stored in the REQ. Related tickets are linked and their contributions can be found in their respective feature fields.	Traceability between different versions of the feature file (INT), tackling the challenge of version control of requirements in BDD (LR [41]). Living documentation (LR) [22] [21].	Any

Table 6.6: Our BDD process.

6.3 Fulfillment of the Requirements on the Implementation

Our process follows BDD principles, introducing a structured way of working with tests and requirements and establishing clear connections between them. It is aimed to be used for Jira tickets where BDD is suitable. This may not be the case for e.g. internal changes and small updates. However, BDD should be applicable for a majority of requirements for a product such as the CRM-team's, where a UI is central. Below, we discuss how our process fulfills the requirements of our implementation listed in Section 5.3.

Our process leads to more clear requirements [IR2] and better fulfillment of them [IR3], two of the requirements for our implementation. By following BDD principles on how to write requirements, through using for example Gherkin, the requirements will be clearly expressed and documented. Having clear requirements facilitates fulfillment of them since all involved employees know what is expected. It is easy to view if requirements are fulfilled. Tests are based on the requirements and passed tests are therefore equivalent to fulfilled requirements, and test results are displayed directly in Jira. How much more clear and how much better the requirements become after applying our process is not yet determined, but can be evaluated through comparisons of how often requirements have to be clarified and through comparisons of the proportion of fulfilled requirements. The number of fulfilled requirements can be hard to measure in teams at the IFE-department who do not have a definition of done. Instead, surveys can be conducted to collect employees' opinions.

Relevant tests [IR4] is a consequence of applying our process, and a requirement of our implementation. Our process, following the principles of BDD, emphasizes writing tests based on the expected behavior of the application, which naturally leads to creating tests that are directly relevant to the requirements and use cases of the software. When a new

requirement is added, or if one is changed, the step definitions that make up the test have to be changed accordingly, based on suggestions that are provided according to our process. If a requirement is removed, the corresponding test is no longer run, even though it still exists in the code base. Hence, all tests that are run are kept relevant. How much applying our process impacts test relevance can be measured by determining what percentage of requirements have corresponding test cases before and after applying it.

Applying our process results in more test automation [IR1], and automation was a requirement of our process. Step definitions are automatically suggested based on new or altered requirements. The tests are run automatically upon changes in production code, and the test results can be viewed in Jira. Feature files are updated and fetched automatically both within Jira and on the test branch on GitHub. There is also automatic naming and creation of branches on GitHub through the tickets in Jira. How much our process has resulted in more test automation can be measured by determining the test coverage of the automated tests. Since some respondents in our interviews mentioned that creating tests often is deprioritized and moved ahead to following sprints, one could also measure in-sprint automation, i.e. how many tests the developers manage to automate within a sprint before and after applying the process.

Easy adoption [IR6] was a requirement of our process. Our process aims to be natural and easy to adopt while contributing to improved quality. Due to the process being BDD tool agnostic, i.e. not dependent on specific tools, its generalizability is high. However, our knowledge of tools from the literature was beneficial when creating our process, as we knew how various tools could be leveraged when applying the process and what is currently feasible. To meet the requirement that the process should be easy to adopt, we decided that the environment of the process should be the one currently used in the CRM-team, i.e. Jira and Github. We did however not investigate alternatives due to our focus on creating an easily adopted process. How easy the process is to adopt can be measured through user feedback and surveys.

The inclusion of traceability between requirements and tests [IR5] is part of our process. Traceability was found to be of high interest to the interview respondents, especially between tests and requirements. We achieve this both since tests are built on requirements and since the requirements are linked to Jira tickets, which also are connected to test branches. We also achieve traceability by connecting Jira tickets to REQs, and having the possibility to view test results in both of them. Traceability between different versions of feature files is provided. The requirements history can be found in Jira tickets and REQs, as well as in GitHubs version history. The requirement on our implementation regarding traceability was that it should be included, which it is. The proportion of tests that are connected to requirements can be used as a metric for measuring this requirement.

Our process is aimed to be applied within a software development team in general, and is tailored to fit the CRM-team. Therefore, we decided to evaluate the process with them specifically. After creating the process, we decided to get feedback from the CRM-team to investigate what could be improved before potentially applying the process.

6.4 Limitations of Implementation

Most steps of the process are feasible and have been performed by us. However, some of them have limitations. Due to time constraints, we have not investigated these limitations further. In a further iteration of our process, the limitations should be investigated, since an implementation of them would improve the process. The limitations are, together with possible solutions, listed below:

- The highlighting of the current changes can not be done automatically. An alternative is to write the addition in bold to make it differ from the recent scenarios.
- Updates of the feature field of the req issue may overwrite content. This will occur when several tickets are ongoing simultaneously. When the tickets are created, they will retrieve the current version of the feature file. If a ticket is created while another one is already ongoing and not yet finished, only the change from the latest finished ticket will remain in the REQ, since it will overwrite the other version. This issue could preferably be solved by implementing version control. For now, we suggest that there is no more than one ongoing change of a ticket at any time.
- External plugins such as the GitHub plugin for Jira are blocked for IKEA's Jira projects. This could be worked around with webhooks to GCP (Google Cloud Platform) functions together with Jira and GitHub automation.
- Test results can not be viewed directly in the ticket or the REQ. With Cucumber, the results can be retrieved and presented nicely in a separate section in Jira, but not in the tickets or REQs.

Chapter 7
Evaluation

The third and last step of our design research, Evaluation, is outlined in this chapter. We begin with describing the method and results of the evaluative workshop we performed. The results are then summarized and discussed. We finish the chapter with suggestions for a further evaluation.

7.1 Workshop

Our process was evaluated in a workshop session of 45 minutes with the CRM-team. The setup of the workshop is described below, followed by a presentation of the results and a conclusion of them, and a discussion of the performed evaluation.

7.1.1 Method

We presented the process to five members of the CRM-team during the workshop. Two of them were also respondents of our interviews in Section 5.1, one of the two and another of the five participants were included in meetings described in Section 5.2. The remaining two were not previously included in our design research activities.

We had an open discussion during the workshop, so the participants were allowed to ask questions, discuss and compare our process with how they work currently. Afterward, the participants were asked to fill out a small anonymous survey that contained questions about the process. Nine questions required answers put on a linear scale from 1 to 5, where 1 meant *Completely disagree* and 5 meant *Completely agree*. These questions were derived from the requirements of our implementation, Table 7.1 presents which of the requirements IR1-IR6 each question relates to. The final three questions were free text and touched on benefits, negatives and general thoughts about the process. The questions are outlined in table 7.1.

#	Question	Scale	Requirement
1	The process seems difficult to adapt to.	1-5	IR6
2	The process would improve traceability between requirements and test cases.	1-5	IR5
3	The process would improve quality assurance in the team.	1-5	IR3
4	The process would help ensure that requirements are fulfilled.	1-5	IR3
5	The process would improve my understanding of requirements.	1-5	IR2
6	The process would improve the testing in the team.	1-5	IR1, IR4
7	The process would improve the test relevance in the team.	1-5	IR4
8	I would not have to change much in my way of working to be able to adapt to the process.	1-5	IR6
9	I would want the team to adapt to the process.	1-5	All
10	What is good about the process?	Free text	
11	How could the process be improved?	Free text	
12	Do you want to add anything else?	Free text	

Table 7.1: Survey to evaluate the process.

7.1.2 Results

The results of the workshop are divided into two parts based on their nature; free text answers and linear scale answers.

Free text answers

All participants answered at least one of the three free text questions. Their responses are summarized below under each corresponding question. The participants are labeled from P1 to P4 based on the order of submission of the survey.

What is good about the process?
P2, P3 and P4 wrote that the process would improve the clarity, readability and traceability of requirements and test cases. P3 argued that it adapts BDD well and would improve the quality assurance of their product. P4 mentioned the benefit of requirements as documentation in the process, as well as having a great automation potential: *"[...] In a perfect world where all the step definitions for the system were in place, this would ensure that tests can be generated on the fly as requirements are added or changed. Today the process is very cumbersome, as our tester would need to go through each affected test and correct it."*.

How could the process be improved?
P2 saw an improvement in having the linkage between the requirement to task in the opposite direction as well, i.e. to propagate changes from the requirement to all linked tasks.

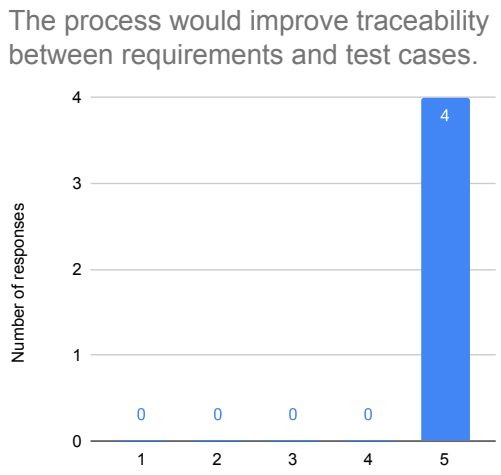
P4 wrote that it would be fantastic if the feature file is automatically sent from Jira to git when a branch is created, and if step definitions could automatically be created. P4 continued by noting that stakeholders often prefer a low bar for adaptation, expressing that they typically prefer crafting requirements according to their preferences, thereby complicating adherence to a predefined structure. However, the participant had a potential solution to the problem: *"One of our engineers said it could be worth it to investigate the possibility of having a generative AI analyze the requirements produced by the stakeholder, and have it link it up to existing step definitions."*. The participant also mentioned the ability to generate a report with the results and automatically deprecate old requirements in Jira as potential improvements to the process.

P3 was skeptical, not of the process itself, but of implementing it: *"Not really a critique of the process, but rather that it would require a large overhaul of the ways of working. It feels like it would take a lot of work to get in place."*

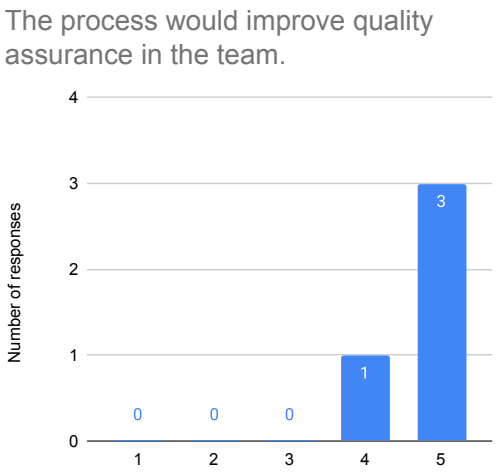
Do you want to add anything else?
P4 wrote that they are optimistic about our solution. P1 emphasized the importance of having a process like ours in place while allowing for agile flexibility: *"I would say we need a process like this to assure the quality of our service. Especially so for established features which should not break when development is continued. I think it has its place, but experimentation and agility is important to keep in mind, so perhaps it should not be used all the time (especially for proof of concepts and experimental implementations)."*

Linear scale answers

The answers to the linear scale questions are presented in Figures 7.1, 7.2, 7.3 and 7.4. The horizontal axis represents the answers put on a scale from 1 to 5, where 1 meant *Completely disagree* and 5 meant *Completely agree*. The vertical axis indicates how many respondents selected a certain number on the scale.



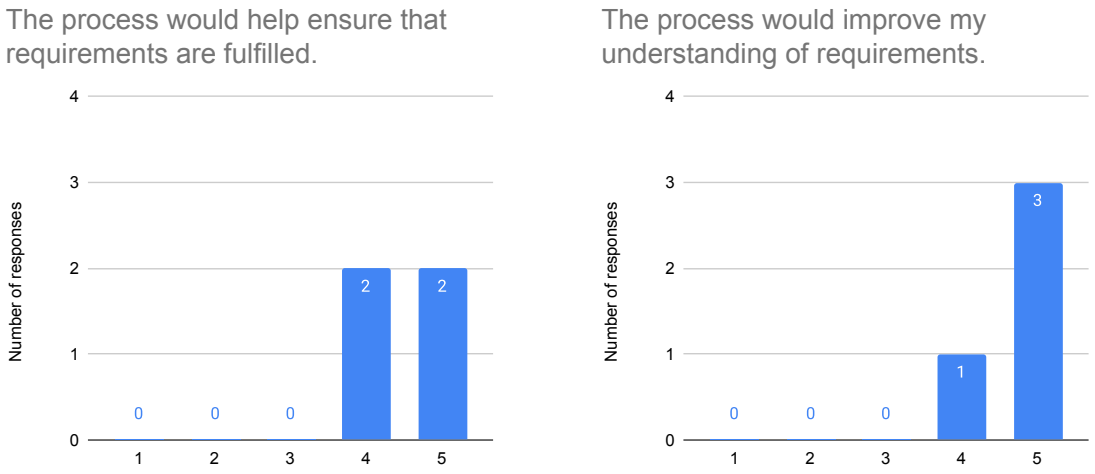
(a) Question 2



(b) Question 3

Figure 7.1: Survey questions 2 and 3.

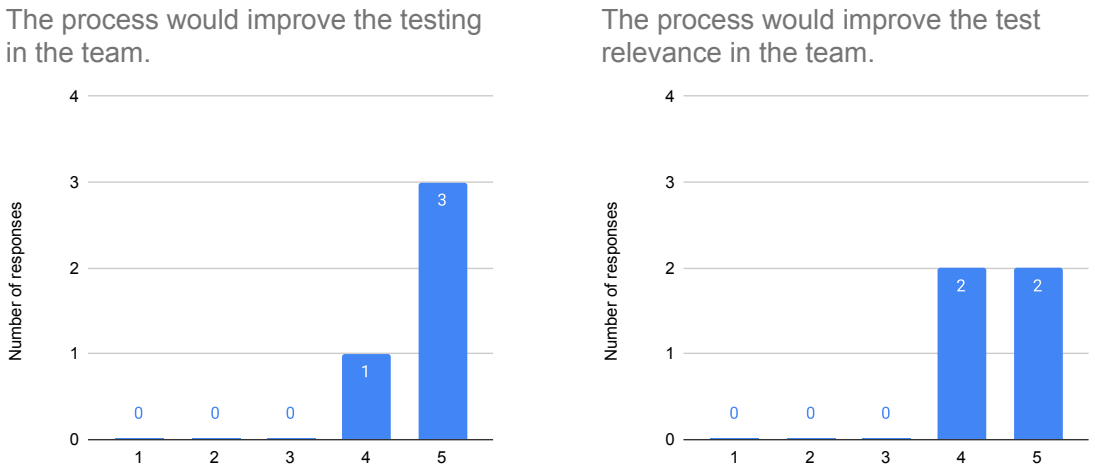
As seen in Figure 7.1a, all survey respondents agreed completely that the process would improve traceability between requirements and test cases [IR5]. The respondents also believed that quality assurance would be improved by adopting the process [IR3] according to Figure 7.1b.



(a) Question 4 **(b)** Question 5

Figure 7.2: Survey questions 4 and 5.

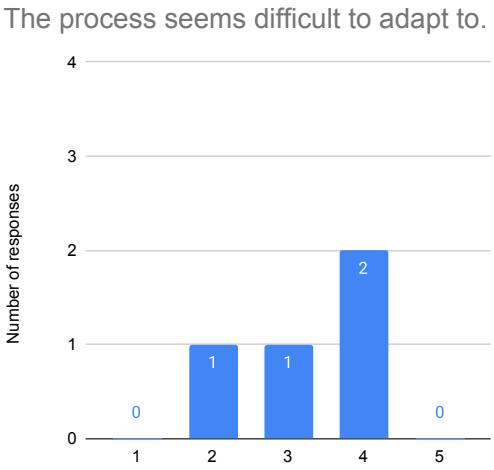
The results of Figure 7.2a indicate that all respondents believe that the process would help ensure that requirements are fulfilled [IR3]. They also strongly believe that their understanding of requirements would be improved when using the process [IR2], as illustrated in Figure 7.2b.



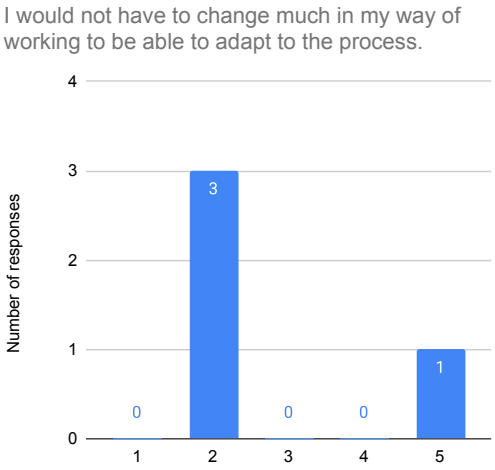
(a) Question 6 **(b)** Question 7

Figure 7.3: Survey questions 6 and 7.

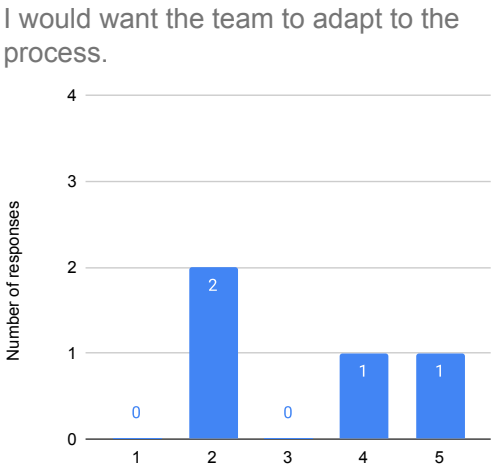
The respondents answered that both testing in general and test relevance would be improved by adapting to the process [IR1] [IR4]. The answers are presented in Figures 7.3a and 7.3b.



(a) Question 1



(b) Question 8



(c) Question 9

Figure 7.4: Survey questions 1, 8 and 9.

There were different opinions on whether the process seemed difficult to adapt to [IR6] as seen in Figure 7.4a, and most answered that they would have to change their way of working in order to adapt to the process [IR6], presented in Figure 7.4b. Figure 7.4c shows that half of the respondents would want their team to adapt to the process, while the other half were negative towards adopting it.

7.1.3 Summary and Discussion of Results

Half of the respondents had a skeptical attitude toward adapting to the process, and some answered that they did not want the CRM-team to adapt to the process. However, all respondents answered that the process could lead to the benefits we asked about and were prominent in the ambitions of the interview respondents.

Improvements mentioned by the participants are valuable when considering further iter-

ations of our process. However, a few of the mentioned improvements were already included in the process. These improvements were to automatically send the feature file from Jira to git and automatically create step definitions. The rest of them could act as a foundation for improved versions of our process.

A skepticism towards applying the process due to the time and effort required could be seen in the linear scale answers of the survey. This finding aligns with results from our literature review, where *steep learning curve* and *might be expensive to implement* are two of the presented challenges. It is reasonable to assume that there exists a conjunction between the skepticism of the survey respondents towards adopting the process and the answers saying that the process seems difficult to adapt to as well as the ones stating that the participants would have to change much in their way of working to be able to adapt to the process. Other reasons behind the skepticism are not covered by the linear scale answers, but one participant emphasized the required work to adapt the process. The respondents' answers indicate that easy adoption is more important than achieving the benefits we asked about in the survey. All respondents answered that adapting the process would imply the benefits we asked about, but there was still skepticism towards adopting it. This suggests that improved instances of the process should show further consideration for a low threshold when introducing the process.

We did not perform further evaluation due to time constraints. Results from the workshop can however be considered as starting points for what to dive deeper into in a further evaluation.

7.2 Further Evaluation

The process was evaluated in a workshop presented in Section 7.1 held with developers and the tester in the CRM-team. Our workshop can work as a reference for how the process could be experienced in a team as well as a collection of opinions on a theoretical level. Due to the constrained time, we could not test the process in practice within the team. It may however not be viewed as a complete evaluation. A more rigorous evaluation did not fit into this thesis but is described in this section and can be conducted in future work.

In future evaluations, the process as a whole should be applied in practice to a team. It could be either the CRM-team or any other team at the IFE-department, depending on resources. Venable et al. present a framework for how the evaluation of a design science process could be done [46]. They do not state for how long the evaluation phase should last, but we have used their evaluation as a benchmark. The average duration of the type of projects the framework was evaluated in was three months. This is equivalent to six sprints in the CRM-team. Before these six sprints of evaluation, the before-metrics mentioned in Section 6.3 related to the requirements on the implementation in Section 5.3 should be measured during two sprints as the team continues working as presently. Thereafter, during the evaluative six sprints, the team should work according to the process. The researchers then take measurements to compare with the previous state to gain an indication of how much the requirements that were set for our implementation have been improved. The members of the team should then be interviewed about their experiences. Observations of the requirements, tests, and production code, along with conducting interviews, should also be performed to give the researchers insights which can further indicate whether the requirements of the im-

plementation are fulfilled and help improve the process in future iterations.

Venable et al. list three goals of evaluation design. These are *rigor*, *efficiency* and *ethics*, and we will use them to describe and motivate how future evaluation of our proposed process can be performed.

Rigor
Efficacy is one aspect, meaning that it has to be the implementation itself that leads to observed improvement. Other, independent factors may not be the reason for the improvement. Another aspect is effectiveness, the implementation must work in practice. To evaluate the implementation in terms of rigor, both aspects need to be considered. However, that may not be possible if the test period is relatively long. On the other hand, it would be hard to evaluate the efficacy if the period of applying the process were too short. Six sprints should be enough to avoid collision with other changes within the team and a reasonable period for the team to avoid other significant changes. When the test period occurs can affect the effectiveness. Different phases of the work within the team adopting the process may come with different challenges. The CRM-team's product is currently live and they have a consistent flow of incoming requirements, where BDD could be used. While that part is suitable for applying the process, one constraint at the moment is the current limitations with the process, which need to be fixed in order to actually enable all steps.

Efficiency
The second goal listed by Venable et al. is that the evaluation should work despite, or minimize, constraints of resources. The team must allocate some time to start working on the process. To make the test period as smooth as possible, the researchers should prepare the team well before starting the evaluation by informing them about the process. The test period should also not be too long, both due to the rigor goal and to be careful with people's time. As long as the previously described issue with the team's Jira connection to GitHub is solved, resources other than time should not be a barrier.

Ethics
The evaluation may not cause any unnecessary risks to animals, people, organizations or the public. Implementing the process as it is described in Section 6.2 requires GitHub to be integrated into Jira, which may expose the organization. This is currently not possible due to safety restrictions. The benefits must be weighed against the risks before allowing the integration and using it in the evaluation. Risks regarding animals, people and the public should not be possible.

Chapter 8
Discussion

In this chapter, we discuss our research questions RQ1 and RQ2. We also discuss some validity threats of the three steps of our design research, focusing on the undertaking of the activities. Thereafter, the generalizability and ethics are covered. The chapter is rounded off with a section describing future work.

8.1 What is the current state of software testing at the IFE-department at IKEA?

(a) How relevant are the tests?

The tests of the teams at the IFE-department are not relevant enough. Testing is often not prioritized in sprints, according to the interview results. Irrelevant and too few tests lead to issues related to quality assurance and system stability. The current testing state must be improved to fulfill interview respondents' needs and wishes since many of them currently are unsatisfied. However, there exist ambitions to improve the testing. On a team level, ambitions for testing include implementing UI tests, leveraging Gherkin, and end-to-end tests, with a desire for more automation.

(b) How are requirements ensured to be fulfilled?

The process for ensuring that requirements are fulfilled differs between teams at the IFE-department. Some teams use various forms of quality assurance to make sure that their requirements are fulfilled. Testing, peer reviews, feedback from users, monitoring tools, and dashboards are leveraged, according to respondents. Most teams also have a definition of done in place. Some respondents viewed a finished ticket, met requirements, or fulfilled acceptance criteria in Jira as being done. Others had more loose descriptions, saying require-

ments are fulfilled when the problem is solved or the functionality works well enough. Some said there is flexibility in the definitions depending on the task at hand.

Teams without a definition of done or any quality assurance lack proper ways to ensure requirements are fulfilled. Respondents viewed its importance differently, and not everyone had considered how they make sure that their requirements are fulfilled or do not even have requirements for their product in place. Having a definition of done was regarded as superfluous by a respondent and by others it was regarded as unnecessary since people had different opinions on its purpose.

The issues that exist with requirement handling probably make it difficult to fulfill requirements. Challenges related to connecting requirements with testing and communicating them properly make the purpose of the requirements vague. The requirements themselves are often not planned adequately, resulting in unclear requirements that often lead to misunderstandings. Hence, the requirements have to be changed often and require a lot of unnecessary communication and time for several stakeholders. Combined with emphasizing delivery speed over quality, it is difficult to ensure that requirements are fulfilled and that the product is functioning as expected.

Improving requirement handling and how they are ensured to be fulfilled across all teams in the IFE-department is essential. Requirements should be in place to prioritize the customer and ensure its satisfaction, respondents in our interviews said. It is relied upon to avoid things breaking, while also encouraging developers to understand the needs of the users.

8.2 How can BDD improve the state of software testing at the IFE-department at IKEA?

The learning curve, difficulty in writing test cases due to its structured nature and potential impact on productivity are acknowledged as challenges associated with BDD in both our interviews and the literature review. Challenges related to resistance to change and the perceived difficulty of adapting to the suggested BDD process are highlighted in our evaluation. However, our results indicate that our process and BDD provide a lot of improvements that respondents in the interviews are seeking.

(a) Can BDD help the IFE-department to have more relevant tests?

Implementing BDD at IKEA can lead to more relevant tests and generally enhanced testing processes. This conclusion is derived from both the problem conceptualization and the literature review, where we learned about needs of the employees and what BDD could contribute, as well as from our process evaluation, where we investigated one instance of BDD.

Generally, BDD practices emphasize aligning tests with requirements and user behaviors, increasing the relevance of tests as the tests act as executable requirements. BDD also improves the organization of tests due to its structured nature of requirements that are interlinked.

Test automation is incorporated in BDD and a key aspect of test relevance since it reduces work and keeps tests up to date. This implies that using BDD would be beneficial for the test relevance. Implementing automation was an ambition of a lot of respondents in our interviews and already used in the CRM-team.

The tester of the CRM-team is leveraging Gherkin when writing UI tests to automate their testing workflow. This indicates that they already experience some of the benefits of BDD, and are open to establishing it more to increase the test relevance. The CRM-team also uses automation in GitHub Actions to trigger the execution of unit tests, as we learned from our meetings and observations.

Our BDD process leverage automation which reduces time spent on testing. It should eliminate pushing the implementation of tests forward to the next sprint, as they are incorporated into the daily workflow of all stakeholders. Implementing our process would improve the testing including the test relevance within the CRM-team, according to the workshop evaluation.

(b) Can BDD help the IFE-department in ensuring that requirements are fulfilled?

Establishing BDD at IKEA can lead to enhanced quality assurance, ensuring better fulfillment of requirements. This conclusion is derived using results from all steps of our design research.

BDD involves all stakeholders, including Product Owners (POs), in the requirement process, improving collaboration and communication leading to a better understanding of system behavior, according to the studied literature. Interview respondents highlighted the importance of involving POs more to improve requirement handling. The Test OKR believes BDD is central to achieving better collaboration and communication across the IFE-department. In teams, our BDD process fosters collaboration at every step, ensuring everyone is aligned. Workshop evaluations indicate that participants believe our process enhances their understanding of requirements, which in turn ensures they can be fulfilled.

Traceability between requirements and tests is fundamental in BDD for verifying requirement fulfillment. Traceability was a focus of improvement according to respondents in our interviews, who also believed BDD could enhance it. Literature also highlights traceability as a key benefit of BDD, since requirements serve as acceptance criteria, with tests directly derived from them. By leveraging tools currently used within IKEA like Jira and GitHub, our process enhances traceability, enabling easy tracking of requirements through test outcomes and related production and test branches on GitHub. Visibility of test results in Jira ensures that everyone remains updated on requirement fulfillment status. Workshop evaluation results further support the belief that our process enhances traceability between tests and requirements.

Improved quality of requirements, a benefit of BDD seen in the literature review, can lead to fulfillment by ensuring that the requirements accurately reflect the needs and expectations of stakeholders. When requirements are well-defined, it becomes easier for development teams to understand and implement them correctly, a wish that was mentioned in our interviews. This reduces the likelihood of misinterpretation or misunderstandings, ultimately resulting in a higher probability of meeting stakeholder expectations and achieving the desired outcomes. Additionally, clear and high-quality requirements facilitate more effective testing and validation processes, allowing teams to verify that the delivered solutions

align with the specified requirements, thus enhancing fulfillment. Ensuring a standard of requirements across the teams in the IFE-department, just like the Test OKR aims to do with BDD, will also facilitate easier collaboration. Workshop evaluation results suggest that participants see potential in our suggested BDD process to improve quality assurance and ensure requirements fulfillment, although some anticipate significant changes in their current way of working, just like the interview results indicated.

8.3 Threats of Validity

For each step of our design research, we have identified a few validity threats that we discuss in this section. They should be considered if our work is applied or continued.

Altering how many respondents, who we chose to interview and which questions we asked could have improved our problem conceptualization. We chose to interview respondents from at least five of the 19 IFE-department teams, ensuring a roughly equal number from each team to balance coverage. Sixteen respondents were deemed sufficient to identify trends and draw representative conclusions for the entire department. However, interviewing more engineering managers (EM) could have led to better insights, as they are involved in several teams and already have a good understanding of what works and what does not in different teams. We also realized our questions were mainly oriented toward developers rather than product owners (PO), leading to some difficulties for POs in answering. Separate questions for POs and developers could have provided better insights, leveraging their respective expertise.

We split longer answers from respondents into smaller codes when analyzing the interview data. One consequence of this was that we lost context of the answers. For example, recording *Do not use sprints* instead of *Do not use sprints because of ...* did not entail a context that could help us assess whether it was a positive, neutral or negative statement. Being more specific would have increased the workload when grouping the codes into categories and themes, but could have made some of the results more meaningful. Another consequence of the split codes was that we did not conder causality between them. For example, if a respondent said that they found their current testing perfect and also that they were not interested in using BDD, a possible conclusion to draw is that the respondent does not want to spend time on improving something that they consider to be flawless. If we had connected the codes instead of treating them individually, it could have resulted in interesting conclusions. However, there would be a risk of drawing conclusions where no causality exists.

In our literature review, a validity threat is that our search was narrow and did not consider alternative solutions to BDD. The search terms could have been broader to include more articles. For example, *test** could have been added after another **OR**, as a narrow search risked that interesting articles were left out. However, we did receive a sufficient amount of articles with different perspectives and focuses in our search. Another aspect of our choice of search terms is that we explicitly searched for articles concerning BDD, meaning we did not read about alternative solutions. Since our research questions are based on BDD, and IKEA had an interest in it, other methods were out of our scope despite what they could have contributed.

Another validity threat is that our evaluation was rather small. Due to time constraints, we could not perform a more comprehensive evaluation. The workshop does however constitute an important first evaluation, which could be utilized both to improve the process and

for designing future evaluations. To collect the feedback from the workshop participants, we conducted a survey. Unfortunately, one of the five participants did not respond. One person constitutes a relatively large percentage of the total in this case, and one more respondent could have added valuable information and contributed to a more extensive result. We do nevertheless consider the results representative for the CRM-team.

8.4 Generalizability

The scope of our research questions is the IFE-department at IKEA and the design research acitivites had different perspectives. The generalizability is however high for both RQ1 and RQ2, and the results can be adapted to different levels of detail. Inside IKEA, we focused on the whole IFE-department in the interviews in the Problem Conceptualization, Chapter 5. We believe that the respondents were chosen so that they represented the whole department. Because of this, the results apply to most, if not all, teams within the IFE-department. The results are more substantial than if we had included only the CRM-team due to the number of respondents as well as their distribution between teams.

Our implementation is designed with the CRM-team in mind. We used interview results as input since they represent the department in which the CRM-team is included. Combined with team-specific results from the meetings and observations and findings from academia retrieved in our literature review, the implementation input had a suitable diversity.

All participants in the evaluation workshop were members of the CRM-team. Their response does not only represent their team's attitudes towards our process but also indicates how the IFE-department would find it. The similarities in the interview answers between teams in the IFE-department indicate that different teams experience similar issues regarding testing and requirement handling, denoting that they would benefit from similar solutions.

Apart from suiting the CRM-team and the IFE-department, we believe that the process can be applied to teams in other departments at IKEA also working with software engineering, as long as they work similarly to teams within the IFE-department. The literature review is not connected to the IFE-department and is therefore relevant to other departments. The interview results also have high generalizability and the respondents' opinions can conceivably be consistent with opinions of employees from other departments similar to the IFE-department.

The majority of our contribution can also be used in contexts outside of IKEA. Results from the Problem Conceptualization, Chapter 5, are specific to IKEA but were found to be similar to other environments when compared to the literature review results. The results of the literature review are not related to IKEA and can be applied in any relevant setting. Our process could also be used outside of IKEA. Assuming that the test and requirement state are similar for other companies operating in the software development domain, our process has a high generalizability. However, it was designed to fit the CRM-team, and is therefore influenced by them through the meetings and observations results. If the assumption of similar states is false, the generalizability may be lowered. Our process is however based upon an extensive literature review where other processes were used as inspiration; therefore, it could be abstracted to suit the needs of teams working in similar fashions (without explicitly using Jira and GitHub).

8.5 Ethics

We stored sensitive data, mainly collected from the interviews, meetings and observations, securely. All documents were created with IKEA accounts to keep the information within the organization. By asking for interview respondents' consent and explaining our intention with the data, we made sure that they were informed of the data handling and knew that they could withdraw their participation whenever they wanted. Combined with letting the respondents know that answers would be anonymized, we could make the respondents feel comfortable sharing information that would be valuable for us. However, since some groups of our respondents were quite few, such as the POs, their anonymity could be threatened. Therefore, we did not present detailed information about the respondents to protect their anonymity. This also applies to the respondents of the workshop evaluation survey, since the number of respondents was small.

8.6 Future Work

While this study has provided valuable insights into the current state of software testing and requirements at IKEA and the potential impact of BDD, there remain several routes for future research and improvement. Addressing these areas could further enhance our process and its applicability at IKEA and other companies in the software development domain.

- The limitations of the implementation brought up in Section 6.4 should be considered and fixed.
- A more substantial evaluation is suggested in Section 7.2. According to it, a team would work according to our process for six sprints before being interviewed about the experience.
- Over time, maintainability of the BDD requirements will become an important aspect. Several articles of Chapter 3, Related Work, have suggested and evaluated viable solutions that can be investigated.
- Ensuring standardized BDD requirements in terms of language is key. Several articles of Chapter 3, Related Work, have suggested solutions such as graphical interfaces and ontologies which can be further explored.

Chapter 9
Conclusion

Testing has been shown to be of great importance. Through interviews with employees representing a whole department at IKEA and meetings and observations focused on one of the teams at the department, we have discovered that testing is found to have a significant role in creating trust for a product and detecting errors instantly. However, a common problem mentioned by interview respondents was that they had too few tests and that their tests were irrelevant. Emphasizing the necessity of tests further, testing was found to be used to validate the fulfillment of requirements. With tests representing requirements focused on user behavior, passed tests are equal to fulfilled requirements, which in turn are equal to satisfied stakeholders.

We have performed a study using design research to explore and improve mentioned issues. It consisted of three steps: Problem Conceptualization, Solution Design and Implementation, and Evaluation. To achieve a state with relevant tests and thereby fulfill requirements, Behavior-driven Development (BDD) was instantiated. We proposed a process based on BDD, tailored to solve the current issues within teams at IKEA.

Following our process could, despite a revealed skepticism, lead to valuable benefits, according to an evaluative workshop we conducted. The potential benefits that the process specifically was intended to contribute were based on previous activities included in our design research. They were mentioned by interview respondents or in meetings we had with the CRM-team as wanted benefits or expressed as issues that had to be solved. These issues were listed as benefits from processes found in the literature review, indicating that the issues could be solved. The benefits were: improved traceability between requirements and test cases, improved quality assurance, helped to ensure fulfilled requirements, improved understanding of requirements, improved testing and improved test relevance. We asked the workshop participants how well they agreed with statements that claimed that the process implied said benefits and the result showed that all respondents agreed that the process would entail all mentioned benefits. Additional benefits, such as that requirements constitute documentation and the automation potential of the process, were also mentioned in the evaluation.

Achieving the mentioned benefits could improve software teams way of working signifi-

cantly. Implementing our process could result in better collaboration between stakeholders, which is of the greatest importance, both in terms of earned resources and established trust for the product. Further evaluation to determine the practical use of our process is suggested in our report.

References

[1] Tefo Sekgweleo and Tiko Iyamu. "Understanding the factors that influence software testing through moments of translation". In: *Journal of Systems and Information Technology* 24.3 (Jan. 2022), pp. 202–220. ISSN: 1328-7265. URL: https://doi.org/10.1108/JSIT-07-2021-0125.

[2] Hisham M. Abushama, Hanaa Altigani Alassam, and Fatin A. Elhaj. "The effect of Test-Driven Development and Behavior-Driven Development on Project Success Factors: A Systematic Literature Review Based Study". In: *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*. Khartoum, Sudan: IEEE, Feb. 2021, pp. 1–9. ISBN: 978-1-72819-111-9. URL: https://doi.org/10.1109/ICCCEEE49695.2021.9429593.

[3] Rareş Cristea, Mihail Feraru, and Ciprian Paduraru. "Building blocks for IoT testing: a benchmark of IoT apps and a functional testing framework". In: *Proceedings of the 4th International Workshop on Software Engineering Research and Practice for the IoT*. Pittsburgh Pennsylvania: ACM, May 2022, pp. 25–32. ISBN: 978-1-4503-9332-4. URL: https://doi.org/10.1145/3528227.3528568.

[4] Chun-Feng Liao et al. "Toward A Service Platform for Developing Smart Contracts on Blockchain in BDD and TDD Styles". In: *2017 IEEE 10th Conference on Service-Oriented Computing and Applications (SOCA)*. Kanazawa: IEEE, Nov. 2017, pp. 133–140. ISBN: 978-1-5386-1326-9. URL: https://doi.org/10.1109/SOCA.2017.26.

[5] Ciprian Paduraru, Miruna Paduraru, and Alin Stefanescu. "RiverGame - a game testing tool using artificial intelligence". In: *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. ISSN: 2159-4848. Apr. 2022, pp. 422–432. URL: https://doi.org/10.1109/ICST53961.2022.00048.

[6] Maria Gerliane Cavalcante and José Iranildo Sales. "The Behavior Driven Development Applied to the Software Quality Test:" in: *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*. ISSN: 2166-0727. June 2019, pp. 1–4. URL: https://doi.org/10.23919/CISTI.2019.8760965.

[7] Kent Beck. *Test Driven Development: By Example*. Google-Books-ID: zNnPEAAAQBAJ. Addison-Wesley Professional, Mar. 2022. ISBN: 978-0-13-758523-6.

[8] Pedro Calais and Lissa Franzini. "Test-Driven Development Benefits Beyond Design Quality: Flow State and Developer Experience". In: *2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. ISSN: 2832-7632. May 2023, pp. 106–111. URL: https://doi.org/10.1109/ICSE-NIER58687.2023.00025.

[9] Daniel Terhorst-North. *Introducing BDD*. Backup Publisher: Dan North & Associates. Sept. 2006. URL: https://dannorth.net/introducing-bdd/ (visited on 01/28/2024).

[10] Fiorella Zampetti et al. "Demystifying the adoption of behavior-driven development in open source projects". In: *Information and Software Technology* 123 (July 2020), p. 106311. ISSN: 0950-5849. URL: https://doi.org/10.1016/j.infsof.2020.106311.

[11] Leonard Peter Binamungu, Suzanne M. Embury, and Nikolaos Konstantinou. "Maintaining behaviour driven development specifications: Challenges and opportunities". In: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Campobasso: IEEE, Mar. 2018, pp. 175–184. ISBN: 978-1-5386-4969-5. URL: http://doi.org/10.1109/SANER.2018.8330207.

[12] Leon Chemnitz et al. "Towards Code Generation from BDD Test Case Specifications: A Vision". In: *2023 IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN)*. Melbourne, Australia: IEEE, May 2023, pp. 139–144. ISBN: 9798350301137. URL: https://doi.org/10.1109/CAIN58948.2023.00031.

[13] Carlos Solis and Xiaofeng Wang. "A Study of the Characteristics of Behaviour Driven Development". In: *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*. Oulu, Finland: IEEE, Aug. 2011, pp. 383–387. ISBN: 978-1-4577-1027-8. URL: http://doi.org/10.1109/SEAA.2011.76.

[14] Tannaz Zameni et al. "From BDD Scenarios to Test Case Generation". In: *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. Dublin, Ireland: IEEE, Apr. 2023, pp. 36–44. ISBN: 9798350333350. URL: https://doi.org/10.1109/ICSTW58534.2023.00019.

[15] Gojko Adzic. *Specification by Example, 10 years later*. URL: https://gojko.net/2020/03/17/sbe-10-years.html (visited on 02/12/2024).

[16] Rakesh Kumar Lenka, Srikant Kumar, and Sunakshi Mamgain. "Behavior Driven Development: Tools and Challenges". In: *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. Greater Noida (UP), India: IEEE, Oct. 2018, pp. 1032–1037. ISBN: 978-1-5386-4119-4. URL: https://doi.org/10.1109/ICACCCN.2018.8748595.

[17] Nan Li, Anthony Escalona, and Tariq Kamal. "Skyfire: Model-Based Testing with Cucumber". In: *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. Chicago, IL, USA: IEEE, Apr. 2016, pp. 393–400. ISBN: 978-1-5090-1827-7. URL: https://doi.org/10.1109/ICST.2016.41.

[18] Pedro Lopes de Souza, Wanderley Lopes de Souza, and Luís Ferreira Pires. "ScrumOntoBDD: Agile software development based on scrum, ontologies and behaviour-driven development". In: *Journal of the Brazilian Computer Society* 27.1 (June 2021), p. 10. ISSN: 1678-4804. URL: https://doi.org/10.1186/s13173-021-00114-w.

[19] Nicolas Nascimento et al. “Behavior-Driven Development: A case study on its impacts on agile development teams”. In: *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. Seoul Republic of Korea: ACM, June 2020, pp. 109–116. ISBN: 978-1-4503-7963-2. URL: https://doi.org/10.1145/3387940.3391480.

[20] M.s. Farooq et al. “Behavior Driven Development: A Systematic Literature Review”. In: *IEEE Access, Access, IEEE* 11 (Jan. 2023). Publisher: IEEE, pp. 88008–88024. ISSN: 2169-3536. URL: https://doi.org/10.1109/ACCESS.2023.3302356.

[21] André Scandaroli et al. “Behavior-Driven Development as an Approach to Improve Software Quality and Communication Across Remote Business Stakeholders, Developers and QA: two Case Studies”. In: *2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE)*. May 2019, pp. 105–110. URL: https://doi.org/10.1109/ICGSE.2019.00030.

[22] Natarajan Thamizhiniyan and Pichai Shanmugavadivu. “Behaviour-Driven Development and Metrics Framework for Enhanced Agile Practices in Scrum Teams”. In: *Information and Software Technology* (Mar. 2024), p. 107435. ISSN: 0950-5849. URL: https://doi.org/10.1016/j.infsof.2024.107435.

[23] Mauricio Alferez et al. “Bridging the Gap between Requirements Modeling and Behavior-Driven Development”. In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Munich, Germany: IEEE, Sept. 2019, pp. 239–249. ISBN: 978-1-72812-536-7. URL: https://doi.org/10.1109/MODELS.2019.00008.

[24] Abhimanyu Gupta, Geert Poels, and Palash Bera. “Generating multiple conceptual models from behavior-driven development scenarios”. In: *Data & Knowledge Engineering* 145 (May 2023), p. 102141. ISSN: 0169-023X. URL: https://doi.org/10.1016/j.datak.2023.102141.

[25] Indra Kharisma Raharjana, Fadel Harris, and Army Justitia. “Tool for Generating Behavior-Driven Development Test-Cases”. In: *Journal of Information Systems Engineering and Business Intelligence* 6.1 (Apr. 2020). Number: 1, pp. 27–36. ISSN: 2443-2555. URL: https://doi.org/10.20473/jisebi.6.1.27-36.

[26] Siti Nadiah Hijriyani, Sri Widowati, and Dana Sulistyo Kusumo. “Application of Behavior Driven Development for Validation Testing (School Information System)”. In: *2022 1st International Conference on Software Engineering and Information Technology (ICoSEIT)*. Bandung, Indonesia: IEEE, Nov. 2022, pp. 96–101. ISBN: 978-1-66547-303-3. URL: https://doi.org/10.1109/ICoSEIT55604.2022.10030005.

[27] Nitish Patkar et al. “Interactive Behavior-driven Development: a Low-code Perspective”. In: *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. Fukuoka, Japan: IEEE, Oct. 2021, pp. 128–137. ISBN: 978-1-66542-484-4. URL: https://doi.org/10.1109/MODELS-C53483.2021.00024.

[28] Daniel Lubke and Tammo Van Lessen. “Modeling Test Cases in BPMN for Behavior-Driven Development”. In: *IEEE Software* 33.5 (Sept. 2016), pp. 15–21. ISSN: 0740-7459, 1937-4194. URL: https://doi.org/10.1109/MS.2016.117.

[29] Leonard Peter Binamungu, Suzanne M. Embury, and Nikolaos Konstantinou. "Detecting duplicate examples in behaviour driven development specifications". In: *2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests (VST)*. Mar. 2018, pp. 6–10. URL: https://doi.org/10.1109/VST.2018.8327149.

[30] Mohsin Irshad, Jürgen Börstler, and Kai Petersen. "Supporting refactoring of BDD specifications—An empirical study". In: *Information and Software Technology* 141 (Jan. 2022), p. 106717. ISSN: 09505849. URL: https://doi.org/10.1016/j.infsof.2021.106717.

[31] Aidan Z.H. Yang, Daniel Alencar Da Costa, and Ying Zou. "Predicting Co-Changes between Functionality Specifications and Source Code in Behavior Driven Development". In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. Montreal, QC, Canada: IEEE, May 2019, pp. 534–544. ISBN: 978-1-72813-412-3. URL: https://doi.org/10.1109/MSR.2019.00080.

[32] Melanie Diepenbeck et al. "Towards automatic scenario generation from coverage information". In: *2013 8th International Workshop on Automation of Software Test (AST)*. San Francisco, CA, USA: IEEE, May 2013, pp. 82–88. ISBN: 978-1-4673-6161-3. URL: https://doi.org/10.1109/IWAST.2013.6595796.

[33] Mazedur Rahman and Jerry Gao. "A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development". In: *2015 IEEE Symposium on Service-Oriented System Engineering*. San Francisco Bay, CA, USA: IEEE, Mar. 2015, pp. 321–325. ISBN: 978-1-4799-8356-8. URL: https://doi.org/10.1109/SOSE.2015.55.

[34] Álvaro Carrera, Carlos A. Iglesias, and Mercedes Garijo. "Beast methodology: An agile testing methodology for multi-agent systems based on behaviour driven development". In: *Information Systems Frontiers* 16.2 (Apr. 2014). Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 2 Publisher: Springer US, pp. 169–182. ISSN: 1572-9419. URL: https://doi.org/10.1007/s10796-013-9438-5.

[35] Kristina Säfsten and Maria Gustavsson. *Forskningsmetodik för ingenjörer och andra problemlösare*. Andra upplagan. Studentlitteratur Ab, 2019.

[36] Claes Wohlin and Per Runeson. "Guiding the selection of research methodology in industry–academia collaboration in software engineering". In: *Information and Software Technology* 140 (Dec. 2021), p. 106678. ISSN: 09505849. URL: https://doi.org/10.1016/j.infsof.2021.106678.

[37] Per Runeson, Emelie Engström, and Margaret-Anne Storey. "The Design Science Paradigm as a Frame for Empirical Software Engineering". In: *Contemporary Empirical Methods in Software Engineering*. Ed. by Michael Felderer and Guilherme Horta Travassos. Cham: Springer International Publishing, 2020, pp. 127–147. ISBN: 978-3-030-32489-6. URL: https://doi.org/10.1007/978-3-030-32489-6_5.

[38] Per Runeson and Martin Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical Software Engineering* 14.2 (Apr. 1, 2009), pp. 131–164. ISSN: 1573-7616. URL: https://doi.org/10.1007/s10664-008-9102-8.

[39] Gerry Larsson Aida Alvinius Anders Borglund. *Tematisk analys: Din handbok till fascinerande vetenskap*. Studentlitteratur, 2024. ISBN: 978-91-44-15102-1.

[40] Barbara Kitchenham and Stuart Charters. "Guidelines for performing Systematic Literature Reviews in Software Engineering". In: *Technical Report EBSE 2007-001, Keele University and Durham University Joint Report* 2.3 (Jan. 2007). URL: https://www.researchgate.net/publication/302924724_Guidelines_for_performing_Systematic_Literature_Reviews_in_Software_Engineering (visited on 03/15/2024).

[41] Mohsin Irshad, Ricardo Britto, and Kai Petersen. "Adapting Behavior Driven Development (BDD) for large-scale software systems". In: *Journal of Systems and Software* 177 (July 2021), p. 110944. ISSN: 01641212. URL: https://doi.org/10.1016/j.jss.2021.110944.

[42] Andrei Contan, Liviu Miclea, and Catalin Dehelean. "Automated testing framework development based on social interaction and communication principles". In: *2017 14th International Conference on Engineering of Modern Electric Systems (EMES)*. Oradea, Romania: IEEE, June 2017, pp. 136–139. ISBN: 978-1-5090-6073-3. URL: https://doi.org/10.1109/EMES.2017.7980399.

[43] Thiago Rocha Silva, Jean-Luc Hak, and Marco Winckler. "A Behavior-Based Ontology for Supporting Automated Assessment of Interactive Systems". In: *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*. San Diego, CA, USA: IEEE, 2017, pp. 250–257. ISBN: 978-1-5090-4284-5. URL: https://doi.org/10.1109/ICSC.2017.73.

[44] Thiago Silva, Jean-Luc Hak, and Marco Winckler. "A Formal Ontology for Describing Interactive Behaviors and Supporting Automated Testing on User Interfaces". In: *International Journal of Semantic Computing* 11 (Dec. 2017), pp. 513–539. URL: https://doi.org/10.1142/S1793351X17400219.

[45] Thiago Rocha Silva, Marco Winckler, and Cédric Bach. "Evaluating the usage of predefined interactive behaviors for writing user stories: an empirical study with potential product owners". In: *Cognition, Technology & Work* 22.3 (Aug. 2020), pp. 437–457. ISSN: 1435-5566. URL: https://doi.org/10.1007/s10111-019-00566-3.

[46] John Venable, Jan Pries-Heje, and Richard Baskerville. "A Comprehensive Framework for Evaluation in Design Science Research". In: *Design Science Research in Information Systems. Advances in Theory and Practice*. Ed. by Ken Peffers, Marcus Rothenberger, and Bill Kuechler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 423–438. ISBN: 978-3-642-29863-9. URL: https://doi.org/10.1007/978-3-642-29863-9_31.

Appendices

Appendix A
Popular Science Summary

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2024-05-31

EXAMENSARBETE Exploring Behavior-Driven Development at IKEA using Design Research
STUDENTER Annie Börjesson, David Jobrant
HANDLEDARE Emelie Engström (LTH), Andreas Trattner (IKEA)
EXAMINATOR Per Runeson (LTH)

Hur får man nöjdare användare genom tydligare krav och relevanta tester?

POPULÄRVETENSKAPLIG SAMMANFATTNING **Annie Börjesson, David Jobrant**

Modern mjukvaruutveckling kräver tydliga krav och relevanta tester för att säkerställa funktionalitet och därmed nöjda användare. Vi har undersökt metodologin behavior-driven development (BDD) och nått slutsatsen att den bidrar till detta, samtidigt som samarbete och kommunikation förbättras.

Vi alla vet hur frustrerande det är när en hemsida inte laddar, eller hur stressad man blir när man inte kan logga in för att köpa biljetter till sin favoritartists konsert. Misströsta ej, det finns en lösning som bidrar till att du slipper dessa problem!

Precis som man testar bilar för att garantera att de är tillräckligt säkra för att köras innan de släpps på marknaden behöver man också testa mjukvaruprodukter för att se till att allt fungerar som det ska innan man släpper dem till användare. För att testningen i sin tur ska fungera som tänkt behövs ett strukturerat sätt att arbeta med tester, där missförstånd och otydliga krav undviks.

I BDD skrivs tester för att säkerställa att användares krav uppfylls, istället för att testa att varje del av koden fungerar. På så sätt kan man enklare se till att användarnas förväntningar möts och därmed få nöjdare användare. Detta gör man genom att skriva krav på ett tydligt format utifrån hur en användare använder produkten. Ett exempel kan ses i figuren till höger.

Vi har identifierat förbättringsområden inom testning och kravhantering på IKEA. Gedigna intervjuer med medarbetare har visat att de bland annat önskar tydligare krav och fler (gärna automatiserade) tester. Kombinerat med resultat från en omfattande litteraturstudie har vi tagit fram en process som följer BDDs principer. Den är tänkt att följas av medarbetare inom olika roller och ämnar till att underlätta både för skaparna och användarna av produkten. Utvärdering av processen bekräftar att BDD bidrar till de förbättringsområden medarbetarna ser.

Scenario:	Uppdatering av varukorg
Given	Jag handlar på IKEAs webbplats
When	Jag lägger till två produkter i varukorgen och öppnar den
Then	Varukorgen ska visa rätt produkter och totalpris.

Vårt arbete har flera olika bidrag. Dels medför undersökningen av IKEAs nuvarande förbättringsområden ökad förståelse bland medarbetare. Dessutom bidrar resultat från litteraturstudien till allmän kunskap om BDD. Vår process kan appliceras i team med liknande förbättringsområden, både innan- och utanför IKEA.

Våra resultat har stora likheter med resultaten från andra studier som undersöker BDD. Det pekar på att upplevelser och åsikter av BDD generellt sett är oberoende av kontexten.